VSB TECHNICAL UNIVERSITY OF OSTRAVA | FACULTY OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Emotion Recognition by the Facial Expressions using Machine Learning

Rozpoznání emocí na základě výrazu tváře pomocí strojového učení

Erik Moravčík

Bachelor Thesis

Supervisor: Sebastián Basterrech, PhD.

Ostrava, 2021

Abstrakt

Byť schopný rozpoznať emócie je kľúčové pre medzi-ľudskú komunikáciu. Táto vlastnosť však chýba pri komunikácii medzi človekom a počítačom. S nedávnym pokrom v hlbokom učení sa vyskytlo veľké množstvo verejne dostupných data setov na účeli rozponávania emócií na základe výrazu tváre. Metódy hlbokého učenia sa následkom toho stali pre tento problém populárne, najmä však konvolučné neuronové siete. V tejto práci prezentujeme dve rôzne architektúry využívajúce jednu konvolučnú neuronovú sieť pre rozponávanie základných ľudských emócií. Okrem toho, prezentujeme aj architektúru založenú na rozhodovacom strome, kde jednotlivé uzly predstavujú konvolučné neuronové siete. Architektúry ktoré prezentujeme sú trénované na známom data sete určenom pre rozpoznávanie emócií na základe výrazu tváre (nazývaný FER2013). Použitím modelu s jednou konvolučnou neuronovou sieťou založenou na VGG architektúre sme dosiahli výsledky porovnateľné s dostupnými výsledkami v odbornej literatúre pre data set FER2013. Ďalej sme otestovali výkonnosť tohto modelu a zároveň aj jeho generalizačné schopnosti na externom data sete. Okrem toho, aplikujeme aj metódy preneseného učenia, aby sme mohli znovu využiť znalosti nadobudnuté pomocou učenia na FER2013 data sete. Vytvoríme nový model, ktorý bude tieto znalosti využívať a otestujeme ich na externom data sete (nazývanom CK+). Využitím tohto modelu sme taktiež dosiahli výsledky na úrovni dostupnej odbornej literatúry pre data set CK+.

Klíčová slova

Hlboké Učenie; Konvolučné Neuronové Siete; Rozpoznávanie Emócií; Prenesené Učenie

Abstract

Being able to recognize emotions is a key component in communication between humans. This component however, is missing in human-machine communication. The recent advancements in Deep Learning (DL) brought with it many public data sets for Facial Expression Recognition (FER). Consequently, DL approaches became popular for emotion recognition, especially Convolutional Neural Networks (CNNs). In this work we present two different architectures utilizing single CNN for predicting basic human emotions. Furthermore, we present an architecture based on Decision Tree classifier, where the decision nodes are CNN models. Presented architectures are trained on a well-known FER data set (identified under the name of FER2013). We achieved state-of-the-art performance using a single CNN model based on the VGG architecture on this data set. We then evaluate performance of our model and test its generalization abilities over an external data set. In addition, we apply Transfer Learning methodology to reuse knowledge assimilated over FER2013 data set. We create a new model, that will be utilizing this transferred knowledge and test it over an external data set (identified under the name of CK+). Employing this new model, we achieved state-of-the-art performance on the CK+ data set.

Keywords

Deep Learning; Convolutional Neural Network; Emotion Recognition; Facial Expression Recognition; Transfer Learning

Acknowledgement

I would like to thank my supervisor Sebastián Basterrech for his guidance, patience and invaluable expertise that helped me during this project. I would like to also thank my sister Lenka, for her indispensable insights and my girlfriend Nikola for her love and support.

Contents

List of symbols and abbreviations

ANN	–	Artificial Neural Network
AU	–	Action Unit
CNN	–	Convolutional Neural Network
DL	–	Deep Learning
DT	–	Decision Tree
FER	–	Facial Expression Recognition
LBP	–	Local Binary Pattern
ML	–	Machine Learning
NN	–	Neural Network
ReLU	–	Rectified Linear Unit
SGD	–	Stochastic Gradient Descent
SVM	–	Support Vector Machine
TL	–	Transfer Learning

List of Figures

List of Tables

Chapter 1

Introduction

Emotions play an important role in inter-human communication. In Conceptual Act Theory, an area of Psychology, emotions such as anger, disgust or happiness emerge from representations of sensations from inside and outside the body and from previous knowledge [1]. Sensation from inside the body (interoceptive sensations) are known as *affect*. They are provoked by activities from peripheral nervous system, neurochemical/hormonal system, and from smooth and skeletal muscles [1]. Sensations from outside the body (exteroceptive sensations) are made up from visual signals, auditory signals, etc. Another factor that impacts emotions is named *concept knowledge* that is understood as the knowledge given by previous experiences about specific feelings [1]. Other models define emotions as domain-specific and that they are produced by specific neural structures [2]. There is empirical evidence showing a relationship between emotions and facial expressions in adults [3]. Ekman presented a theory of facial expressions where he identified the following basic emotions: happiness, surprise, disgust, sadness, anger, and fear. Most of the research has been done over that set of basic emotions [4]. Humans are able to detect changes in each others emotional states and act or react accordingly. However, this aspect is not fully integrated in human-machine communication, where basic exchanges are messages that do not consider emotional information from the human user [5]. It would be helpful to develop system that could integrate emotional information, that could in turn be used for improving human-machine experience, as well as for interpreting human behavior. There are several ways, how to detect emotional states. One of them is to monitor internal body signals, such as neural activity, pupil size, beating heart, and many others medical signals. Another approach is to detect emotions from physical signals, such as speech, gesture, posture and facial expressions [6]. Facial expressions have become essential and one of the most common ways for human beings to express their emotions and emotional states [7].

With the huge progress made in the last few years on pattern recognition over images, and the strong relationship between facial expressions and emotions, there has been a raising interest in developing automatic systems for recognizing emotions from images of human faces. With the dawn of Deep Learning (DL) era and with a lot of emotion recognition competitions, many public data sets

for Facial Expression Recognition (FER) aroused and DL based approaches have been adopted more and more [8]. Mostly Convolutional Neural Networks (CNNs), that allow us to obtain automatic features utilizing convolution filters. When such filters are applied on images, various features can be extracted, such as edges and corners [9]. These features can be then combined to form more specific features, for instance human mouth or nose. In CNNs, filters are considered as learnable parameters, this allows them to capture peculiarities of image data and learn more robust features, in contrast to traditional methods, that rely on handpicking features manually. Nevertheless, traditional FER algorithms are still being used to this date and are able to accomplish state-of-the-art performances on some widely used FER data sets [8].

In this work, we apply CNN for recognizing basic human emotions identified by Ekman from images containing human faces [4]. We present two different single CNN architectures, one based on more traditional LeNet-5 architecture [9]. The second one is inspired by VGG architecture [10]. Furthermore, we present architecture based on Decision Tree (DT), where the decision nodes are CNN models. In addition, we evaluate generalization performance of proposed architectures in a new environment using Transfer Learning (TL) principles. TL will allow us to preserve knowledge learnt in the first place and reuse this knowledge in the domain of the new data set. For the purposes of our work, we used probably the two most popular FER data sets, FER2013 [11] and CK+ [12]. FER2013 is a large database of facial expression images, for this reason we utilize it for the training of our presented architectures. We use CK+ data set for the purposes of evaluation and for applying TL. In order of experimental replicability and reproducibility the full code and other sources are available for the scientific community in a public repository[1].

The remainder of this thesis is organized as follows. Chapter 2 presents an overview of recent methods used for FER. Theoretical background regarding NNs in general and also CNNs is presented in Chapter 3. In Chapter 4, we present a brief overview of available data sets on FER area, and we discuss with more details the data sets used along the thesis. In chapter 5, we describe procedures we use for preprocessing of images, optimization algorithms and specify architectures of presented approaches. Experimental results are provided in Chapter 6.

[1]Full code and other sources: `https://github.com/erik-mor/Emotion-recognition`

Chapter 2

State Of The Art

In this chapter we present recent approaches and methods used for FER. Chapter is divided into 2 sections, one describing DL based approaches, using mainly deep CNNs. The second section presents more traditional ML approaches.

2.1 Deep learning based facial emotion recognition methods

In [13, 14, 15] authors employ an ensemble of CNNs, in which multiple networks are combined and predictions of individual networks are evaluated using voting procedures. In [13] a specific architecture was used for each of the networks in the ensemble, the architecture was composed by five convolution layers and stochastic pooling. Furthermore, instead of averaging the output of individual networks, which is a prevalent way of integrating networks together (voting), they propose a different structure that utilizes weighted ensemble response, treating the weights as learnable parameters. Another approach was studied in [14], where authors demonstrate the use of different architectures for single CNNs. Besides, they apply different pre-processing (normalization, augmentation) in the training images creating different training data sets for each single network. For the ensemble voting they utilize exponentially-weighted response. A novel method called Sample awareness-based personalized was proposed in [15]. Instead of performing voting to choose the classifier for the test sample as in [13, 14], in this method the classifier is selected according to each test sample. They utilize Bayesian learning to assign the optimal classifier out of the ensemble for every individual training sample. Based on this data, new classifier is trained to determine the proper classifier to use for individual test samples.

Sometimes FER models are misled by personal or other characteristics present in the image samples that are not related to emotions. This problem is tackled in [16] and in [17]. Identity-aware CNN is proposed in [16]. Authors implemented two equivalent CNNs with shared weights. On top of softmax layer, used in both CNNs for the purpose of the FER classification, joint contrastive loss is utilized. Contrastive loss is employed to learn features related to identity and other personal

characteristics. Different approach is taken in [17]. Authors propose architecture consisting 2 separate CNNs, outputs of these CNNs are then fed into fully-connected layer as Tandem Facial Expression features. The purpose of the first CNN is for recognizing personal characteristics and features. The second one is used for the FER task. These networks are pre-trained using separate data sets for the respective tasks. After that the whole network is trained only with FER data set.

Novel single CNN model based on the Inception architecture [18] was presented in [19]. For the pre-processing, facial landmark were detected and data augmentation was employed. Authors present results of experiments on seven well-known FER data sets, including FER2013 and CK+ used in our work. They also conducted cross-database experiments to test generalization abilities of proposed architecture. The main difference of this work with ours, is that we present different single CNN architecture, as well as DT of CNNs. Furthermore, on top of evaluation of presented architecture over external data set to test its generalization abilities, we also apply TL techniques.

NN with Relativity Learning was proposed in [20]. Authors use variation of Inception architecture [18]. They utilize improved triplet loss function, in which they map image features to Euclidean space and computed relative distance between features with same and with different facial expression. Based on this distance, gradient was updated, so as to assign more weight to samples that are more difficult to recognize.

In [21] authors develop model for recognizing learning emotions. For learning emotions, there is not that much data collected compared to basic emotions. To tackle this problem they implemented two-phase TL. At first they train model using smaller data sets for basic emotion recognition problem. Then the knowledge is transferred on the model trained with bigger data set (FER2013), again for basic emotion recognition. Finally, from this model knowledge is transferred to model for learning emotion recognition problem. As the architecture they propose Dense_FaceLiveNet architecture, which is in fact, a refined version of FaceLiveNet [22].

2.2 Traditional machine learning facial emotion recognition methods

In [23] authors extract features from active patches with the use of 68 landmark locations. Features are extracted using Local Binary Patterns (LBPs). To get the most useful features, features are combined with the gray values of active pathces. They also employed dimensionality reduction using Principal Component Analysis of gray values of the patches combined with LBP features. For the final classification Softmax layer was used.

Approach utilizing salient facial patches based on novel facial landmark detection method was presented in [24]. Authors employed learning-free facial landmark detection method, to detect eyes, nose, eyebrows and lips. Based on these landmarks, active facial patches were extracted. Rather than using all facial patches, authors used only patches that are necessary for the specific facial expression. From these patches, features were extracted using LBPs. For the classification, Support Vector Machines (SVMs) were used.

In [25] authors propose Local Directional Ternary Pattern method. To each pixel in input image is designated an eight bit code. Robinson compass masks are employed to represent edges shapes. Edge magnitude is used to differentiate between useful and insignificant patterns. Thanks to this, features that are extracted are more useful since extracting is done in the more meaningful regions of the face.

Computational model for real-time emotion recognition based on facial expressions was presented in [26]. Authors use Candide grid to encode geometrical information based on facial landmarks as well as for the tracking of face in the video frames. For the classification, the change in the grid formation between the first frame and the last frame is fed into the SVM that outputs one of the six basic emotions or the Facial Action Units (AUs).

Dynamic texture-based approach for recognizing emotions was proposed in [27]. Authors utilized changes of facial AUs in video frames based on free-from deformations. Instead of classification based on whole videos or just the frames, authors are working with sliding windows with variable sizes, that differ for individual AUs. For the final classification of AUs, Gentle-Boost classifier and Hidden Markov Models are incorporated.

Chapter 3

Convolutional Neural Networks

Neural network (NN) or Artificial Neural Network (ANN) is a computational model inspired by the human brain, in particular, its structure and the way brain processes information from its receptors and handles specific functions, such as cognition, motor control or pattern recognition [28]. Just as human brain, ANN is comprised of large number of interconnected processing units called neurons. Each neuron has a set of input connections (dendrites), bias, and the output (axon). Every input connection has a weight assigned to it, representing the strength of the connection. Visualization of the neuron is shown on Figure 3.1. To determine the output of the neuron, weighed sum of input connection values is computed where the weighting is given, intuitively, by the weight of the connection. The results of this is combined with the bias term and then fed through the activation function, that will regulate the output based on some boundaries [28]. Formally, we can represent this as [29]:

$$z = \sum_j (x_j w_j) + b, \tag{3.1}$$

where b denotes the bias term, x represents the input connection, and w represents the weight of the connection. Sum is over all input connections and weights. Let z be the result of this equation, or the activation of the neuron. It is common to represent output of the neuron as a non-linear function, referred to as activation function. Purpose of non-linearity is to increase computational power of the network. To formally represent this, we can substitute z in the following formula [29]:

$$a = \sigma(z), \tag{3.2}$$

where a represents the non-linear output of the neuron, σ is the activation function, that has one parameter z, which represents the activation of the neuron.

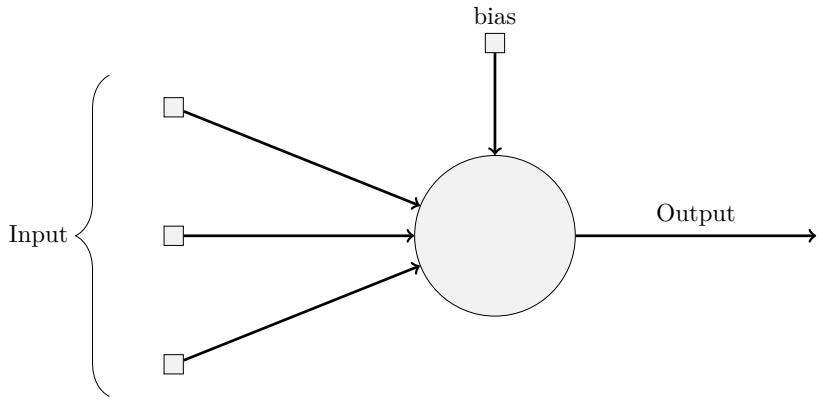


Figure 3.1: Visualization of artificial neuron.

There is a relatively large number of activation functions, for instance sigmoid, tangent, linear, binary, etc. We will take a closer look at two activation functions that we will be using in this work. At first we discuss the softmax activation function, which is excellent choice for the classification tasks with multiple classes, thanks to the fact, that the individual output neurons can represent probability distribution of the present classes [29]. Softmax function is defined as [29]:

$$a_i = \frac{e^{z_i}}{\sum_k e^{z_k}}, \tag{3.3}$$

where a and z are analogous to the formulas (3.1) and (3.2), for the i-th output neuron. The sum in the denominator is over all activations z of neurons in the output layer. Next activation function used in this thesis is Rectified Linear activation function. Unit that employs this activation function is referred to as Rectified Linear Unit (ReLU). ReLU behaves in a way as a linear function, when it outputs exactly what it has on input, except, if it receives a negative input. When it has positive input, learning is consistent and effective, however, if the input is negative the learning stops, since it outputs 0 [30]. ReLU is defined as follows [30]:

$$a = \max(0, z), \tag{3.4}$$

where a and z are again analogous to the formulas (3.2) and (3.1).

Based on neuron organization in the network, we distinguish two different ANN architectures, feedforward networks and recurrent networks. In feedforward network, neurons are arranged as a sequence of successive layers, starting with input layer representing input to the network and last layer representing the output. In between, there can be zero (single-layer feedforward network) or one and more so called hidden layers [28]. The architecture is shown on Figure 3.2. This way, information is forwarded through the network from input layer to the output layer, so as, output from neurons in hidden layers represent the input for the neurons in the next layers. On the other

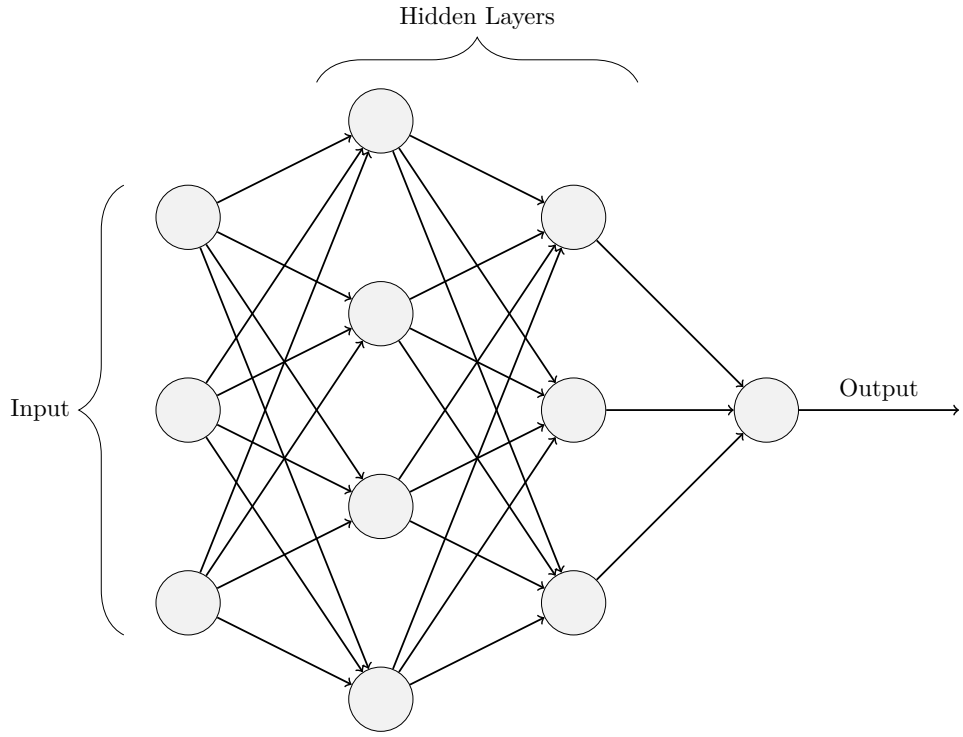


Figure 3.2: Scheme of the feedforward network.

hand, recurrent networks contain so called feedback loop, i.e. output of the neuron can be used as input to the neurons in the previous layers or even to itself (self-feedback loop) [28].

CNNs are special type of standard ANNs, in particular Feedforward networks, that proved to be successful in processing input data organized in a grid structure, for example images [30]. The main distinction between them, is that CNNs are using so called convolution operation in at least one of their hidden layers. This is at the same time one of the most important features of CNNs, thanks to which, the number of parameters can be reduced compared to ANNs, and that in return allows them to perform so well in the field of Computer Vision [30].

3.1 Convolution operation

In mathematics, convolution operation is described as [31]:

> "Convolution is a mathematical operation on two functions (f and g) that produces a third function ($f * g$) that expresses how the shape of one is modified by the other. It is defined as the integral of the product of the two functions after one is reversed and shifted."

The convolution operation is denoted with an asterisk. The formula for convolution operation is as follows [31]:

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\gamma)g(t-\gamma)d\gamma. \tag{3.5}$$

From the above formula can be seen that convolution is defined as weighted average of some function $f(\gamma)$ at the time t, where the weight is defined as $g(-\gamma)$ shifted by time t, and therefore smaller the time higher the weight. This mathematical definition is analogous to the ML, where function f can be thought of as input and the function g as the kernel or filter, later in this thesis, these two terms will be used interchangeably. The final result of convolution operation on our input and kernel, or the output, is sometimes called *feature map*. In the above formula, functions f and g have only one parameter. In reality, since CNNs deal with data organized in a grid structure, that are most of the times represented as two-dimensional matrices, these functions will have two parameters. Therefore we can rewrite our formula in the following way [30]:

$$(f * g)(i, j) = \int_0^m \int_0^n f(m, n)g(i-m, j-n). \tag{3.6}$$

3.2 Convolution operation over images

The convolution on images is performed by sliding the kernel over the input and computing weighted sum at every step [32]. Then, the product of this sum will be added to the output, similarly as in formula (3.6). This process is depicted on Figure 3.3.

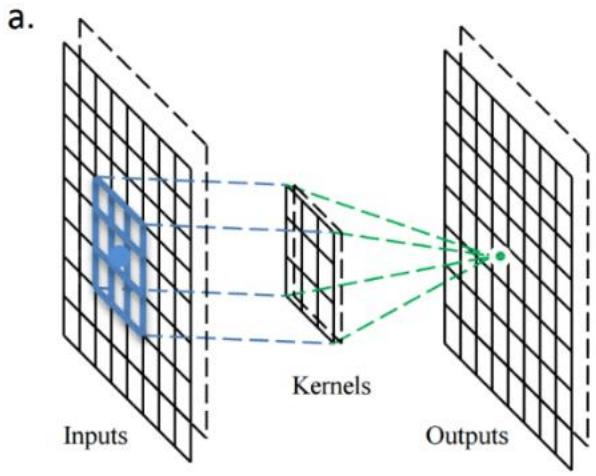


Figure 3.3: One step of the convolution process, i.e. taking weighted sum of input weighted by the kernel and projecting the product to the output [33].

When applied to images, convolutions proved to be successful at detecting different kinds of features [34]. In the case of the first convolution layer the input is some image and we can detect

features such as vertical or horizontal edges. When we go deeper into the network, input here is represent by output or the feature map of preceding layers and we are able to detect more complex features, for example some group of edges that can represent human nose or mouth [34]. On Figure 3.4 we can see this in more detail. First there is 6×6 input image on the left. As higher intensity pixels are transitioning to the lower ones (in form of 10s and 0s) there is strong vertical edge in middle of the image. When we convolve it with 3×3 filter depicted in the middle of the figure, the output will produce 4×4 image on the right side of the figure. On this feature map we can see that the vertical edge from the original image is detected and even highlighted. The output is smaller than the input since we are not padding the input image and the stride of 1 is used, more about stride and padding will be discussed in the next section.

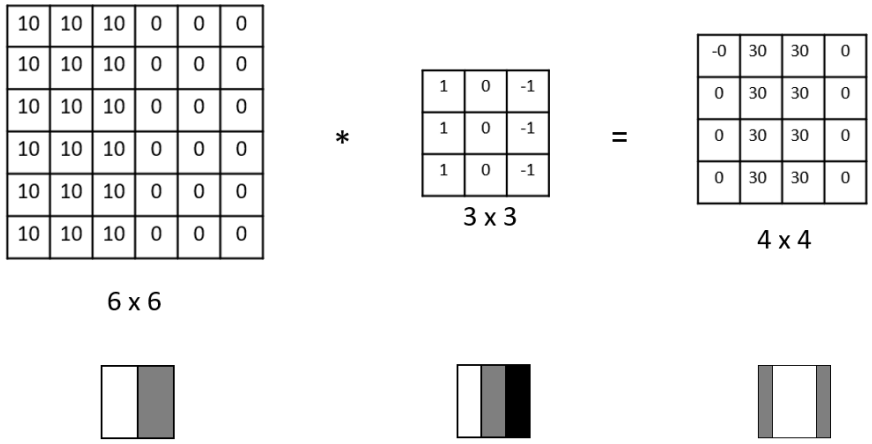
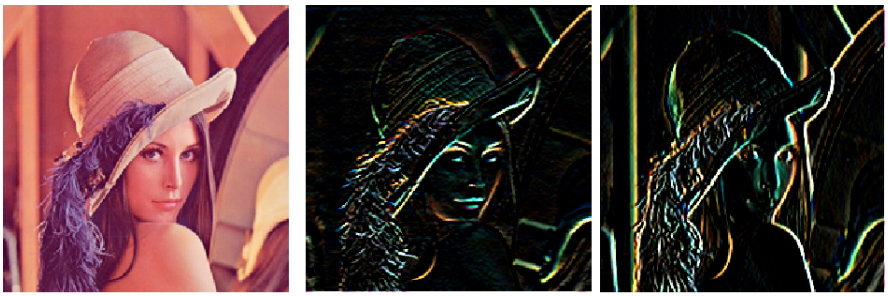


Figure 3.4: Vertical edge detection on simple 6×6 image [35].

Different filters have different properties that are allowing them to detect different kind of features [32]. On Figure 3.5 picture a), we can see more complex input image and using similar approach, applying more popular filter for edge detection, so called Sobel filter $\mathrm{F} = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$, we can see that this simple method is efficient even when input image is more complex and we can detect vertical and horizontal (we transpose filter F, so we get $F^T = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$) edges.

Optimal parameters from which are these filters comprised can be learned through classic gradient based algorithms. This approach is more effective than to handpick these numbers manually [9]. In each layer of the network we can then include several of these filters and each one of them can then recognize different features, from basic edges in the first layers to more complex features in



(a) Lenna (b) Horizontal edge (c) Vertical edge

Figure 3.5: Edge detection showcased on popular Lena image [36].

deeper layers. When we fuel the network with enough images, and fed these images through this sequence of layers, it can be used to recognize different kinds of objects in real word [37].

3.3 Padding and stride operation

From Figure 3.4 is observable that after convolving a 6×6 matrix by 3×3 filter the dimensions of the output image shrinks to 4×4. This number can be reduced even more and it is influenced by parameter called *stride*. In our example, we are using stride of 1, and therefore, when we slide 3×3 filter along the input image with the stride of, we can compute the weighted sum only 4 times before filter will run out of image borders. If the stride of 2 would be used the output dimension will be 2×2, since this time we would run out of borders after the second slide. This is not always wanted quality of the convolution operation, since the dimensions can shrink too much and some information can be lost [37]. Especially, information on the edges and in the corners is thrown away, since filter is processing this part less often than for example some pixels in the middle of the image. One solution to this is to use padding. When padding is used, zeros are added around image so dimension of the former image is increased from $A \times B$ to $A + 1 \times B + 1$ [37]. Thus, after convolution is performed on this image, original dimensions are maintained and thanks to that more hidden layers can be present in the network and no information is lost. In addition to that, pixels around edges have now much more impact on the output as when no padding was used [37]. It is also possible to use more than one layer of zeros around image, however, this is not really common. In practice, there are two most often used options regarding padding. First option, called *valid* padding, is when there is no padding applied to the image, so dimension is reduced, as in our example on Figure 3.4. The second option is called *same* padding. That is, when the input image is padded around, so the output dimension remains same as the input.

3.4 Pooling operation

Another crucial operation used in CNNs is called *pooling*, or sometimes referred to as *sub-sampling*. Pooling is mainly performed after the convolution and its main purpose is to reduce the dimension of the output produced by convolution step to decrease complexity of the representation [30]. Applying pooling can also highlight features detected by the convolution operation [30]. The two most used types of pooling are *max pooling* and *average pooling*. The process of pooling is similar to the convolution operation. It is in fact almost same, in the sense that we slide the filter along input with some stride and produce some output. When pooling is performed, we do not compute weighted sum. Instead, max or average value is taken out of the slice of the input that the filter is currently processing, depending on what variant of the pooling we are using [37]. This value is then placed on the output. This process is depicted on Figure 3.6. Pooling also brings invariance to the network, what basically means that when the input is slightly changed the output of pooling will remain the same, since we always extract max values [30]. This in other words means, that it is not important where some feature is present in the input image. What is important, is the fact that the feature was detected.

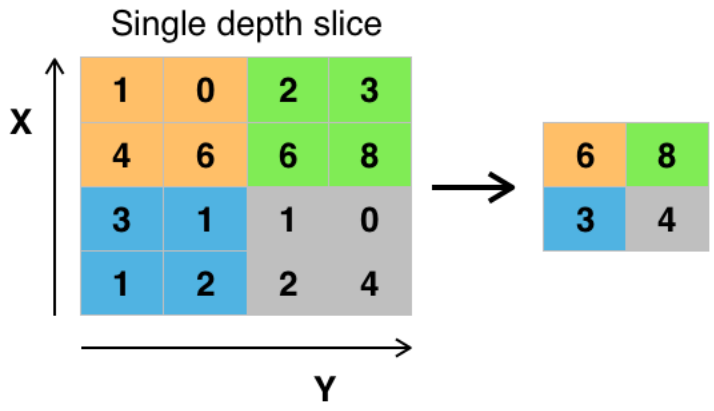


Figure 3.6: Max pooling process with filter of size 2×2 and the stride of 2 [38].

3.5 Optimal architectures

In [9] authors proposed the LeNet-5 network to recognize zip code digits. This work became starting point for further research and experiments in the field of computer vision using CNNs. Until then, the fundamental problem was how to choose the optimal set of input features, which were handpicked manually. In the LeNet-5 network, features are determined by the network itself using convolutions. The network consists of 7 layers, at first there are 2 convolution blocks comprising convolution and pooling. After the convolution blocks comes 2 fully-connected layers followed by the output layer. From this architecture, other popular architectures has been derived. For instance Alex-Net [39],

used in ImageNet LSVRC-2010 contest [40]. Alex-Net implemented more layers and is in general much more complex architecture. Because of this, dropout was also implemented to reduce over-fitting.

In [10] authors propose architecture called VGG. They are improving on the original Alex-Net architecture [39], by rapidly increasing the depth of the network, what is possible because of the utilization of small 3×3 convolution filters across all convolution layers. Furthermore, padding is used for the convolution operation to retain the dimensions of the input. The general architecture is as follows, there are five convolution blocks, that are comprised from one to four convolution layers followed by max pooling layer. After this convolution stack come three fully connected layers and output layer. This approach is in some way contradictory to the Inception architecture [18]. In Inception architecture, authors implement sparse connections between layers with dimension reduction using 1×1 convolutions to reduce the depth. They claimed that increasing the depth may in some cases cause over-fitting. This architecture is inspired by novel approach of network in network [41]. The sparse connections present in the network, form so called Inception module.

3.6 Learning optimal parameters

In the previous sections, we discussed the basics of ANNs, their structure and neuron organization. Later we looked at some different algorithms and features of CNNs, and some common CNN architectures. In this section we will describe how NNs learn, or in other words how to find optimal values for the network parameters (weights and biases) to make adequate predictions. Learning, regarding machines is broadly described as [30]:

> "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."

Task T has a relatively broad meaning and can represent a number of ML tasks, for instance linear regression, machine translation, detection of anomalies and many more. Accuracy or error are most often metrics that we can consider as the performance measure P. For computer program to learn from experience E, large amount of data is needed. This data can be labeled (i.e. each data sample has a label that is signaling to which class this sample belongs), which is often refereed to as *supervised learning* or learning with teacher [30]. Data can be also without label, what is know as *unsupervised learning*, where the algorithm has to make sense of data and find similarities by itself, for instance clustering [30]. In the context of this thesis, task T can be thought of as classification, as we are developing automatic system to recognize human emotions. In other words, we are developing system, that given set of training examples (images), will attempt to find optimal parameters (weights and biases), such that, the output of our system will approximate the true label of the input image. For this purpose we can derive the performance measure P, as a cost function

J, that will represent the error of our approximation. We can formally represent this cost function as follows [29]:

$$J(w, b) = \frac{1}{m} \sum_i (\hat{y} - y)_i^2. \tag{3.7}$$

This is in fact a well know cost function called *quadratic cost function* or *mean squared error*. Parameters w and b are representing weights and biases of the network, m denotes number of training examples, where i is individual training example. The prediction of the system is denoted as $\hat{y}$, and the actual label as y. This function is basically computing an average error for all training examples. Error can be thought of just as Euclidean distance between the prediction $\hat{y}$ and the actual label y. We can see that if $\hat{y}$ is equal to y and thus the prediction is correct, the error is 0. On the other hand, if the distance is bigger the error will be in return bigger too.

To summarize, our goal is to find optimal parameters w and b, such that minimize the cost function J. The most common approach for minimizing the cost function J is to perpetually change parameters w and b based on the gradient of the function J with respect to individual parameters w and b until convergence [29]. At each point of this perpetual update the process consists of two steps. At first, the gradients with respect to parameters w and b (or derivatives) of the cost function J are calculated using the back-propagation algorithm. Back-propagation is efficient algorithm for calculating and evaluating these derivatives and as the name suggests, it propagates the error of the prediction back to the network, layer by layer. This propagated error serves as the means for adjusting individual weights and biases. In general, back-propagation can be summarized as follows [42]:

- Forward propagation of the network activation (3.2), from input through hidden layers to output.
- Calculate the error of the output neurons, as a gradient of the cost function with respect to activation of the output neurons combined with the derivative of output activations.
- Propagate this error backwards into the network, layer by layer.
- Based on this error, derive the gradients for individual parameters w and b in each layer.

After the gradients for parameters w and b are calculated, second step is to update these parameters, so as to minimize the cost function J. How this update is performed depends and the training algorithm, but essentially all of them are inspired by the update of some parameters to minimize some cost function based on the gradient [30]. The most straightforward algorithm based on the gradient is called gradient descent. In gradient descent, at every point we move slightly in the steepest direction to the minimum of the cost function [42]. In the update rule there is parameter η called learning rate that has the role of regulating the speed of the weight update. The learning rate is usually a small number between 0 and 1. Let denote ∇J the gradient vector, i.e. the vector

of all partial derivatives of cost function with respect to individual parameters (weights and biases). We can formally represent this update rule for the weights w as:

$$w^{new} = w^{old} - \eta(\nabla J). \tag{3.8}$$

We are omitting the definition of update for bias b since this update is conceptually identical as for the weights w. In classic gradient descent, update of the parameters is performed after computing gradient for every training example. However, computation of gradient can be costly computationally speaking, especially if there are many hidden layers present in the network. To tackle this problem, iteration of the classic gradient descent algorithm was established, called Stochastic gradient descent (SGD). In SGD, we split the training data into groups, called mini-batches. The weights are updated after computation on every such mini-batch. The results is not as precise as for the whole training data, but computation is much faster and more effective [29]. Another variation of the gradient descent is called momentum, or gradient descent with momentum. The momentum helps speed up the learning, since to update weights it takes into account old gradients. Also, unlike in the SGD, weights are not updated directly using gradient, but using intermediary term v, which is often referred to as velocity, that represents the exponentially weighted average of the previous gradients [30]. Velocity update is performed as follows [30]:

$$v^{new} = \beta v^{old} - \eta(\nabla J), \tag{3.9}$$

where β denotes hyperparameter, that represents how much importance we give to the previous gradients. Parameter η once again symbolizes the learning rate, and ∇J denotes gradient vector. Using the velocity v computed in (3.9) we can update our weights w as follows [30]:

$$w^{new} = w^{old} + v. \tag{3.10}$$

Recently, popular algorithm named Adam was presented in [43]. Adam stands for adaptive momentum estimation, and is essentially combination of the gradient descent with momentum and RMSprop [44] with few improvements. RMSprop is another popular gradient based algorithm, it has adaptive learning rate for individual weights, so as the gradient is divided by the exponentially weighted moving average of past gradients. As was demonstrated in the original paper, Adam proved to be very efficient and quick, surpassing other popular optimization algorithms.

3.7 Implementation

Theano [45] is a Python library developed for working with mathematical expression, in particular with multi-dimensional arrays, making use of the Numpy library. Theano is one of the oldest ML frameworks started in 2007. Its execution is conveyed as computational graph and can make use of

only single CPU or GPU. The implementation in Theano is low-level, on the other hand, compilation process is quick compared to other frameworks. To make implementation more user-friendly, Theano integrates with high-level wrappers, such as Keras.

Tensorflow [46] is a one of the most popular ML platforms. It offers flexible architecture, where it is easy to build and deploy ML models in the cloud, browser or on mobile device. It also offers a large scale of useful tools and libraries, and has also a large community. Its execution is conveyed similarly as in Theano as a computational graph, but can make use of multiple CPU's or GPU's.

PyTorch [47] is a ML library based on Torch (older ML library based on programming language called Lua). PyTorch is based on Python as the name suggests. It has user-friendly API and as opposed to Tensorflow, utilizes dynamic computational graphs. The advantage of this, is that just fractions of the code can be tested and the implementation can be adjusted on the run. It also offers a large number of handy tools and libraries.

Keras [48] is a python based high level API built on top Tensorflow. Keras offers easy to use interface for ML solutions, specifically for modern DL approaches. The main building blocks of Keras API are models and layers. Sequential model represents the easiest model representation, i.e. the stack of layers. Keras also offers tools for implementing more complicated models or write own model implementations through inheritance. To create a model, at first the structure must be defined, that is, to add individual layers that the model will be comprised of. There are several types of layers, for instance Dense layer (regular fully-connected layer, as on Figure 3.2), or Convolutional layer. Each layer has its own set of parameters, but in general it is number of units (in Convolutional layer number it is the number of filters) and the activation function. When the layers are defined, the next thing to do is to define optimization parameters, such as cost function, optimization algorithm (SGD, Adam, RMSprop), learning rate, and metrics that will be useful during the training process (accuracy or loss). Once this configuration is done, training phase can begin. For the training, training data and optionally validation data needs to be specified, and also number of epochs, mini batch size and even some callback functions that may be useful during the training.

Keras Tuner [49] is a powerful library designed to perform search for the best set of network hyperparameters. There are several configurations to choose from regrading the searching algorithm, such as Hyperband [50], that uses adaptive resource allocation or the classic Random Search. Similarly as when training model is created using Keras API, when utilizing Tuner, such model is also needed. In contrast, this model will be used for the tuning of the hyper-parameters or the hypertuning, and is named accordingly - hypermodel. Along with hypermodel, search space for individual hyper-parameters needs to be defined. Afterwards, the search itself is performed in specific number of iterations or trials. This search depends on the specified objective of the search, i.e. what to optimize (accuracy, loss).

Chapter 4

Description Of Data Sets

In this chapter we present an overview of available data sets regarding FER. We present the benchmark data for problems on the area of supervised learning (i.e. data with labels), and also for unsupervised learning (i.e. unlabelled data).

4.1 Review of available benchmark data sets

1. Data-set Aff-wild, containing 300 videos where each frame is annotated with regard to valence and arousal. Videos are captured in the wild (i.e. uncontrolled environment), mostly YouTube videos. Data set is available at `http://bit.ly/aff-wild`.

2. Data-set Aff-wild-2 which is extended version of the previous benchmark data set aff-wild. Data set contains 539 videos, each one is annotated for 7 basic emotions. Source `http://bit.ly/aff-wil2`.

3. The extended Cohn-Kanade data set (CK+), containing 327 video sequences of 118 subjects, each annotated for 6 basic emotions and contempt. This is one of two selected data sets for the purposes of our work, more thorough description is available below. Data-set is available at `http://bit.ly/ckplusextend`.

4. Facial Expression Recognition 2013 Dataset (FER2013), containing over 34000 images for 7 basic emotions included in the data set. This is also on of the selected data sets for the purposes of this thesis and more information is available below. Available at `http://bit.ly/fer2013`.

5. Data-set CelebA, containing 202599 images of 10177 people taken in uncontrolled manner, therefore there can be background noise, different postures, etc. Images are annotated with 40 different attributes (e.g. glasses, hat, smile, mustache, etc.) and also 5 facial landmark locations. Source `http://bit.ly/celebfacesA`.

6. FFHQ data set, consists of 70000 facial images with 2 possible size parameters - 1024×1024 or 128×128. Images taken from Flickr, and containing a considerable amount of background noise, varying poses, different ethnicity and illumination. Source `http://bit.ly/ffhqflickr`.

7. Google facial expression data set containing around 50000 triplets with box points and annotations specifying which two faces are the most similar in the triplet based on facial expression. Source `http://bit.ly/googlefec`.

8. Labeled Faces in the Wild data set, consisting of 13233 images of 5749 people. Images are centered with face cropped using Viola Jones face detector. Size of the images is 250×250 pixels. Images available at `http://bit.ly/labfacw`.

9. Real and Fake detection. Data-set containing 2041 high quality images intended for classification into two categories - fake and real. Images also divided into easy and hard selections. Source `http://bit.ly/realfakedet`.

10. Simpsons faces data set, consisting of 19800 images of The Simpsons characters - one half are cropped facial images other half is simplified version using face recognition algorithm (only skin and eyes). Available at `http://bit.ly/simpface`.

11. Tufts Face Database, consisting over 10000 images containing 7 image modalities. Source `http://bit.ly/tuftsface`.

12. UMDFace data set, consists of 367888 face annotations of 8277 subjects with bounding boxes around the face, pose (i.e., yaw, pitch, and roll) and locations of 21 landmarks. Also included more than 3.7 million annotated video frames from over 22000 videos of 3100 subjects. Source `http://bit.ly/umdface`.

13. UTKFace data set containing 20000 facial images captured in the wild. Images are centered, cropped for the faces and with 68 facial landmarks. Images are also labelled by age, gender, and ethnicity. Available at `http://bit.ly/utkface`.

14. Wider Face data set, consists of 2000 to 4000 images, labels specify different occasions - parade, meeting picnic, etc. Source `http://bit.ly/widerface`.

15. Yale faces database, consists of 165 Gif images of 15 subjects. Every subject is presented in 11 different facial expressions or configurations, e.g., glasses, happy, normal, wink, etc. Source `http://bit.ly/yaleface`.

16. YouTube Faces With Facial Keypoints data set. Consists of 2200 videos of famous people with up to 6 per each subject. Each video consisting of up to 240 frames with bounding boxes, 2-D landmarks and 3-D landmarks. Source `http://bit.ly/ytfaces`.

4.2 Selected benchmark data sets

As the main data set for the Emotion recognition problem, that is going to be used for the training of our FER system, we selected FER2013 data set [11]. FER2013 is popular data set in the area of FER. It consists of over 34000 images, taken in uncontrolled environment (in the wild), and therefore, there is high variability regarding pose, angle and illumination. On Figure 4.1 are shown example images form data set. Images are of size 48×48 pixels in gray-scale. Images are labeled for 6 basic emotions and neural emotion. In Table 4.1 is shown distribution of images per each class. We can see that data set is not evenly distributed among present classes, which was one of the problems that we had to deal with. The data set is on itself divided into another three groups:

- Training - 28709 images - this group is for the training of the network
- Private Test - 3589 - this group is used as validation data in the training process
- Public Test - also 3589 images - used for the testing of the network

The second selected data set, called CK+ [12] is for the purpose of the final evaluation and TL. This data set is much smaller compared to the FER2013 data set. It consists of 593 video sequences of 123 subjects. Furthermore, only 327 sequences are actually labeled for 6 basic emotions and contempt. Video sequences start at neutral expression, and capture the transition up to the peak expression. Sequences were captured in lab environment, thus all of them are taken from frontal view and configuration for all of them is more or less the same. Publicly available data set taken from `http://bit.ly/ckplusextend`, contains 981 images, where for each one of 327 video sequences, there are three representative images. In our case, we take into account all images from this public data set, but also just the peak facial expression (just one image), therefore we end up with just 327 image samples. Distribution of images among present classes is shown in Table 4.2. On Figure 4.2 we can see example images from the data set. Furthermore, data set is not implicitly divided for training and testing as opposed to FER2013 data set. Common approach is to divide data set into K groups and perform K-fold cross-validation, K is usually 5, 8 or 10 [8]. Data sets differ only in one of the labels, in the FER2013 data set, there is neutral class, in the CK+ data set, there is class contempt. An overview of state-of-the-art results for the two selected benchmark data sets are presented in Table 4.3.

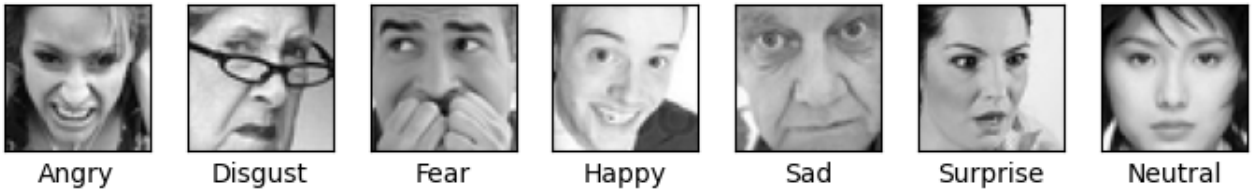


Figure 4.1: Examples of images from FER2013 data set.

Figure 4.2: Examples of images from CK+ data set.

Table 4.1: Distribution of images per class in FER2013 data set.

Class	**No. of images**	**Emotion**
0	4593	Anger
1	547	Disgust
2	5121	Fear
3	8989	Happiness
4	6077	Sadness
5	4002	Surprise
6	6198	Neutral

Table 4.2: Distribution of images per class in the CK+ data set.

Class	**No. of images**	**Emotion**
0	45	Anger
1	59	Disgust
2	25	Fear
3	69	Happiness
4	28	Sadness
5	83	Surprise
6	18	Contempt

Table 4.3: State of the art performance.

Data set	**Accuracy (%)**	**Reference**
CK+	95.37	Meng et al. [16]
CK+	99.3	Li et al. [17]
CK+	93.2	Mollahosseini et al. [19]
FER2013	72	Yu et al. [13]
FER2013	72.72	Kim et al. [14]
FER2013	71.08	Li et al. [15]
FER2013	66.4	Mollahosseini et al. [19]
FER2013	70.6	Yanan et al. [20]
FER2013	70.2	Hung et al. [21]

Chapter 5

Methodology

In this chapter, we describe procedures used for preprocessing of images, and also optimization and regularization techniques used during the training process. We will next define architectures and experimental settings for single CNN model and also for the individual networks in DT. At last, we will introduce utilized TL techniques.

5.1 Experimental settings

5.1.1 Learning data set

Our data set is not evenly distributed among present classes, see the number of sample per class shown in Table 4.1. For the creation of a more balanced learning data set, we applied the following procedure. We arbitrary selected a number that represents number of samples per class, that will be used for each of the emotion classes (let's denote it by M). This number was around 80% of the amount of the samples in the happiness class (the class with the largest amount of samples). Then, we sampled M samples per each class, therefore the number of samples in every class will be equal, but some classes will also contain repeated samples. With this procedure we obtained a balanced data set, then it may improve the performance of our model.

5.1.2 Image preprocessing

We employed data augmentation, which is a way to inflate training data. Data set is enriched with images from original data set, that have certain modification. This will help the model to generalize well and it is also one of the most common practices to prevent over-fitting [39]. Applied data augmentation in our case:

1. image rotation - range for random rotation
2. width and height shift - fraction of total height or width for the random shift

3. horizontal flip - flipping the image horizontally

4. zoom - range for the random zoom

5. shear - shear intensity

Normalization of the learning data set is a common procedure, it helps in the optimization of the NN. Normalized data decreases the problems provoked by neuron saturation and unstable gradient computations. For this reason, we applied min-max normalization to rescale pixel values between 0 and 1. Once the values are rescaled the training process will be more stable.

5.1.3 Optimization algorithms

As a cost function we utilized cross-entropy which is a common metric used for classification problems [29, 51]. To complement cross-entropy cost function, we selected softmax activation function for our output neurons, described in more detail in Section 3.6. Softmax along with cross-entropy cost function is a prevalent combination these days for classification using CNNs [52]. As an activation function in hidden units was used ReLU, described already in Section 3. ReLU is one of the most commonly used activation functions in hidden units in CNNs these days. When ReLU is used, training process in CNNs is much faster and more effective compared to other choices of activation functions [39].

As a optimization algorithm to minimize the cost function we selected Adam, which was already described in Section 3.6. Adam algorithm belongs to one of the most popular choices of gradient based algorithms among ML experts. Furthermore, Adam proved to be effective when used in CNNs, as was demonstrated in the original paper [43]. We used low learning rates of 0.001 and 0.0001 which is suggested by the authors. Also, parameters β_1 and β_2 are at default values of 0.9 and 0.999 respectively, as recommended.

A good practice to reduce over-fitting is to regularize the model [53]. The regularization of the model was done in two ways. At first we were inserting dropout layers. These layers are typically inserted after fully connected layers, or behind convolutional blocks. Dropout layers, as the name suggests, bring dropout into the network, what simply means that some parameters, or neurons, are randomly dropped during the training. This will reduce complexity of the network, and thus, reduce the over-fitting [39]. The next approach in order to regularize the NN is to incorporate penalty term to the cost function, such that, it prevents the weights of the neurons from becoming too big and therefore reducing the complexity of the network [54]. We implemented L1 and L2 regularization. The goal was to find optimal parameters for L1 and L2 in each convolution layer. However, in our case, this process was not efficient.

5.1.4 Performance metrics

To evaluate performance of presented methods, following metrics were used [55]:

- Accuracy: percentage of ratio between samples that were predicted correctly versus number of all samples.
- Precision: ratio between accurately predicted positive samples and number of samples predicted as positive. In other words, if we are for instance predicting images just to one class, let's say happiness, it is number of images accurately predicted as happiness divided by number of all images predicted as happiness, even the wrong ones.
- Recall: ratio between accurately predicted positive samples (we can once again think of this as number of images predicted as happiness) and number of images that are actually positive (i.e. all images labeled as happiness).
- F1 score: combination of precision and recall.

5.2 Single convolutional neural network

In single CNN setting, we used two different architectures, first one (base model) is a variation of the standard LeNet-5 architecture [9]. Second model (final model) is based on VGG architecture [10], that utilizes deeper network with help of 3×3 convolution filters.

5.2.1 Base model

Network architecture for the base model is show in Table 5.1. The network consists of 3 convolution blocks, within each block there is one convolution layer with 3×3 filters, with stride of 1 and same padding. After the convolution layer comes the max pooling layer with 2×2 pool size and stride of 2. After the convolution blocks the output is flattened before being fed through the fully connected layer with 160 units. At last there is output layer with 7 units corresponding to 7 emotion classes in the data set. For the base model, we did not use any regularization techniques nor the data augmentation.

5.2.2 Final model

The architecture for final model is shown in Table 5.2. Network consists of 4 convolutional blocks. Within each block there two convolutional layers followed by batch normalization. After that comes max pooling layer for dimension reduction and also dropout. After this stack of convolution blocks comes flattening layer followed by fully connected layer. We again apply batch normalization and dropout after the fully connected layer. At last there is the output layer with 7 units corresponding to our 7 emotion classes. Since this architecture is much more complex then the base model, we tackle the over-fitting problem in 3 ways [56]. During preprocessing we applied data augmentation [39]. After every convolution and fully connected layer batch normalization was applied, which normalizes the distribution of training examples in the mini-batches, that will in return help with

Table 5.1: Architecture of the base model.

Convolution - 64 Pooling
Convolution - 128 Pooling
Convolution - 160 Pooling
Flatten
Fully Connected - 160
Fully Connected (Output)

the regularization of the network and also with generalization abilities [57]. Thanks to this, we can apply less dropout.

5.3 Decision tree composed of convolutional neural networks

5.3.1 Overview

When predicting on all 7 classes by single CNN architectures, we achieved reasonable performance. Nevertheless, some classes were still problematic. We decided to try different approach - to create DT of CNNs, where each CNN will be working only with some specified part of the data. We can train dedicated models just for these individual parts. We expect that with this approach we can achieve higher performance. In the proposed DT, each leaf node corresponds to one of the emotion classes (shown in Table 4.1). Root node and parent nodes of this tree are represented by CNNs responsible for recognizing only one designated subset of these classes. The scheme for the tree is depicted on Figure 5.1. The main goal is to optimize each network to achieve the best possible performance. When testing, each image will be fed through this tree to get the predicted result. As can be observed from scheme, NN_1 is responsible for recognizing only class 2 (fear). The training data for this model consist of data set split into two evenly distributed parts, one containing images of class 2, and the second one containing all other image classes together. The job of the NN_2 is to split the remaining 6 classes again into two selections. Classes 0 (anger), 4 (sadness) and 6 (neutral) in the first one, and 1 (disgust), 3 (happiness) and 5 (surprise) in the second. These selections are handled by last two CNNs respectively.

5.3.2 Architecture and experimental settings

Every CNN in the DT has similar architecture as final model for single CNN 5.2. Configuration for the individual networks can be seen in Table 5.3. As can be observed from the table, the

Table 5.2: Architecture of the final model.

Convolution - 64
Batch Normalization
Convolution - 64
Batch Normalization
Pooling
Dropout
Convolution - 128
Batch Normalization
Convolution - 128
Batch Normalization
Pooling
Dropout
Convolution - 256
Batch Normalization
Convolution - 256
Batch Normalization
Pooling
Dropout
Convolution - 256
Batch Normalization
Convolution - 256
Batch Normalization
Pooling
Dropout
Flatten
Fully Connected - 256
Batch Normalization
Dropout
Fully Connected (Output)

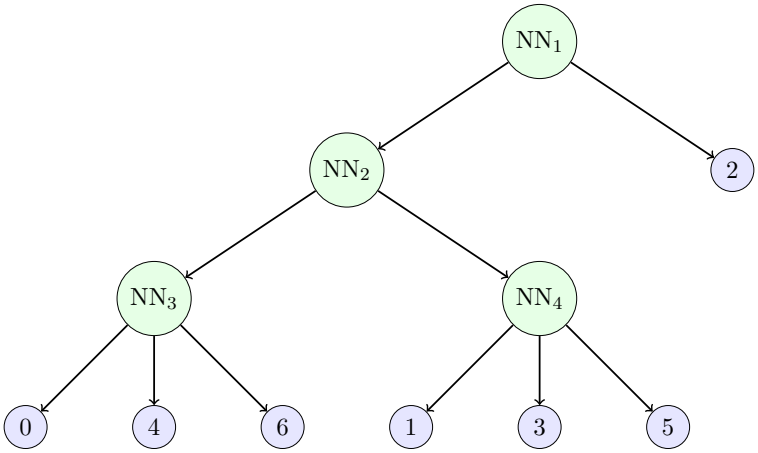


Figure 5.1: Decision tree composed of convolutional neural networks. Leaf nodes (blue) are representing individual emotion classes. Green nodes representing convolutional neural networks.

Table 5.3: Experimental settings for individual convolutional neural networks in the decision tree.

Network	Conv. blocks	Data augmentation					
		Rotation	Width shift	Height shift	Zoom	Shear	Horizontal flip
NN_1	3	-	0.15	0.15	0.15	-	No
NN_2	2	20	0.15	0.15	0.20	0.20	Yes
NN_3	3	40	0.40	0.40	0.40	0.40	Yes
NN_4	2	20	0.15	0.15	0.20	0.20	Yes

difference between networks is only in the number of convolution blocks and the settings for data augmentation.

5.4 Transfer learning

Furthermore, we studied the application of TL principles to the previously presented ML tools. When applying TL, we can retain knowledge learnt in some domain (source domain) and reuse it in another domain (target domain) [58]. The main idea is that, we can benefit from some existing knowledge from source domain in the target domain, similarly, as we humans can apply our existing knowledge in some novel situations. In this work, we trained a model to recognize human emotions based on images from FER2013 data set, which we will refer to as data set 1. In this case, we can think of data set 1 as our source domain. We can transfer the knowledge learnt by this model and based on that create a new model that will be retrained in the new domain, i.e. CK+ data set, or data set 2. As mentioned in [34], in lower layers (closer to input), CNN is able to recognize some low-level *general* features, such as edges and corners. In the successive layers, such features are

merged to form higher level feature detectors, responsible for recognizing much more complicated and more *domain specific* features. Therefore the knowledge that will be transferred in our case is represented by the features learnt by our model, specifically in convolutional layers (features learnt using convolution filters). The goal is to evaluate these features over the external data (data set 2), i.e. our new target domain. The transferred model will be almost the same as our original model 5.2. We will keep already learnt weights and biases in the network and set them as untrainable, i.e. when training, these parameters will not change (we want to retain the features learnt in the training in the source domain). Subsequently, we will replace last fully connected layers with new ones, and initialize weights randomly. When training on new data, weights will be updated only in these layers (what represents the retraining in the target domain). We are not retraining the convolutional layers since they contain the learnt features we want to preserve.

Chapter 6

Experimental Results

In this chapter, we present results of individual architectures for single CNN, as well as results for the DT. Then, we discuss obtained results and select the optimal architecture that will be evaluated in the CK+ data set to test generalization abilities of proposed architecture. In addition, we apply TL methods to improve our model in the new domain of CK+ data set.

6.1 Single convolutional neural network

The base model 5.1 scored accuracy of 95% on training data and 55% on validation data after training for 50 epochs. What indicates, that there is over-fitting problem since our validation accuracy is lacking compered to accuracy on training data. This results is not unexpected, since regularization was not applied. Therefore this model learnt unique quirks and gimmicks of the training data and was not able to generalize well on data that is has never seen before [56]. However, when the regularization was applied, model was unable to learn even on thw training data and achieved sub-optimal performance.

Final model achieved state-of-the-art performance with regard to results shown in Table 4.3 on FER2013 data set, with accuracy of 66%. Other evaluation metrics are shown in Table 6.1. On Figure 6.1 is shown confusion matrix, depicting the results. We can see that some classes, are more problematic then the others, for instance class 2 (fear) with accuracy of 45%.

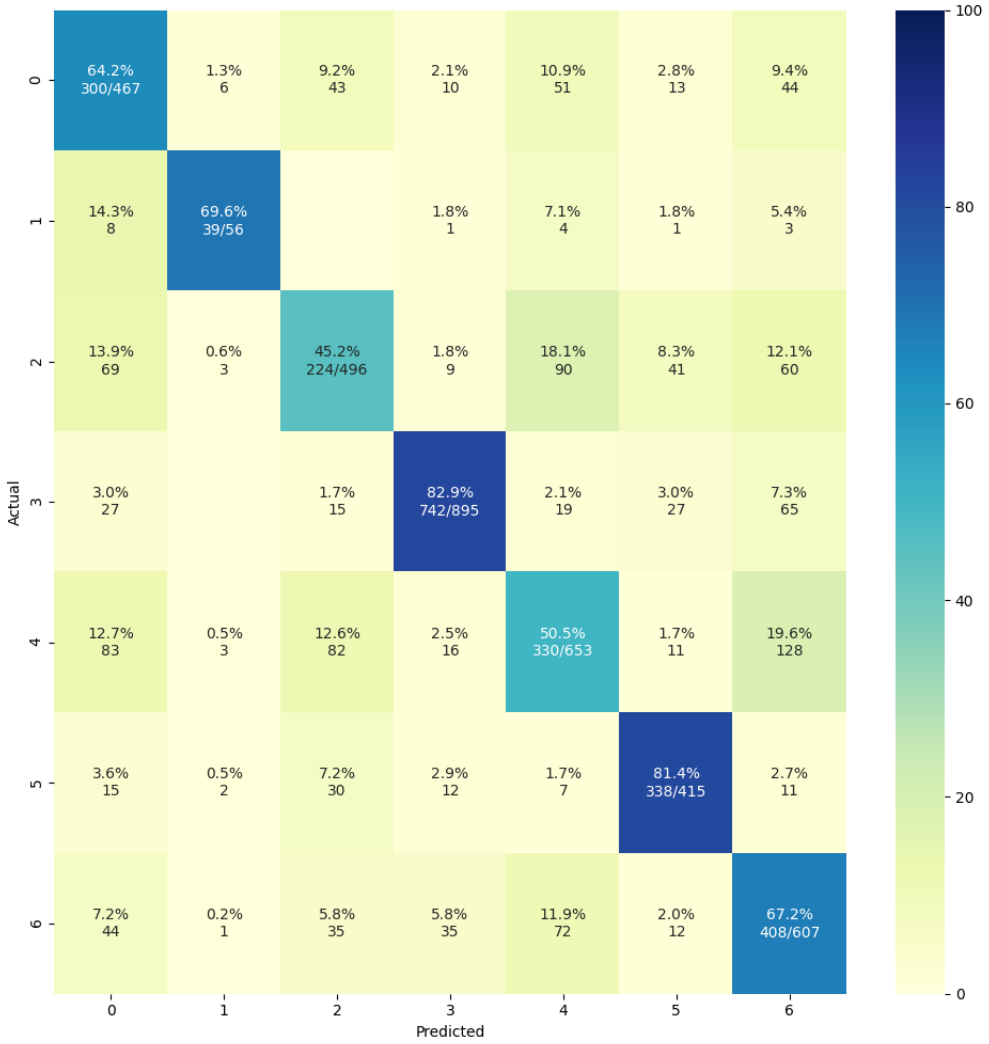


Figure 6.1: Confusion matrix depicting results of the final model. On the Y axis we have the actual labels, ranging from 0 to 6 representing emotion classes. On the X axis, there are predicted labels by the network. For instance position 0 on X and Y axis represents situation, where the actual label for input image is 0 and also the prediction of the network was 0. It is therefore desirable to have higher numbers along the main diagonal.

Table 6.1: Classification report for the final model.

Class	Precision	Recall	f1-score	No. of examples
0	0.55	0.64	0.59	467
1	0.72	0.70	0.71	56
2	0.52	0.45	0.48	496
3	0.90	0.83	0.86	895
4	0.58	0.51	0.54	653
5	0.76	0.81	0.79	415
6	0.57	0.67	0.62	607
Average	0.66	0.66	0.66	3589
Weighted Average	0.67	0.66	0.66	3589

6.2 Decision tree

- NN_1: NN_1 is responsible for recognizing class 2 (fear) vs all other classes. The main motivation behind creating dedicated model just for class 2 was that, the single CNN for recognizing all classes had the biggest problem recognizing it. Results for this model can bee seen in Table 6.2 and confusion matrix 6.2. As can be observed, after dedicating specific model for this class, recognizing it was still problematic. Both precision and recall are low for class 2, that means model was often not correctly predicting class 2 when other classes were the input and also failed to find the actual images of class 2.

- NN_2: The purpose of NN_2 was to split data into two selections, as mentioned in the previous chapter. This model yielded solid results, with accuracy around 85% and both precision and recall are above 80%. Detailed results can be seen in Table 6.3 and on confusion matrix 6.3.

- NN_3: NN_3 is responsible for predicting classes contained in selection 1. This selection contains classes 0, 4 and 6. This model achieved mediocre performance, with accuracy of 67%, precision and accuracy also stayed at around 60%. Detailed metrics can be seen in Table 6.4 and on confusion matrix 6.4.

- NN_4: Analogously to NN_3, NN_4 is responsible for recognizing classes in second selection, specifically 1, 3 and5. This was the best performing network out of all in the DT. Network scored accuracy of 93%, precision and recall were around 90%, only class 0 (in translation class 1 (disgust)) falling behind, as can be observed in Table 6.5 and on confusion matrix 6.5.

Results when testing on the overall structure can bee seen in Table 6.6 and on confusion matrix 6.6. As can be observed, isolating problematic classes did not help the overall performance and

accuracy went down to 54%. On the other hand, accuracy for the isolated class (class 2) did get better, but the difference is insignificant.

Table 6.2: Classification report for NN_1.

Class	**Precision**	**Recall**	**f1-score**	**No. of examples**
0	0.30	0.48	0.37	496
1	0.91	0.82	0.86	3093
Average	0.60	0.65	0.61	3589
Weighted Average	0.82	0.77	0.79	3589

Table 6.3: Classification report for NN_2.

Class	**Precision**	**Recall**	**f1-score**	**No. of examples**
0	0.85	0.88	0.86	1727
1	0.84	0.81	0.82	1366
Average	0.85	0.84	0.84	3093
Weighted Average	0.85	0.85	0.85	3093

Table 6.4: Classification report for NN_3.

Class	**Precision**	**Recall**	**f1-score**	**No. of examples**
0	0.69	0.67	0.68	467
1	0.68	0.60	0.64	653
2	0.64	0.73	0.68	607
Average	0.67	0.67	0.67	1727
Weighted Average	0.67	0.67	0.67	1727

Table 6.5: Classification report for NN_4.

Class	**Precision**	**Recall**	**f1-score**	**No. of examples**
0	0.85	0.84	0.85	56
1	0.96	0.94	0.95	895
2	0.89	0.93	0.91	415
Average	0.90	0.90	0.90	1366
Weighted Average	0.94	0.93	0.94	1366

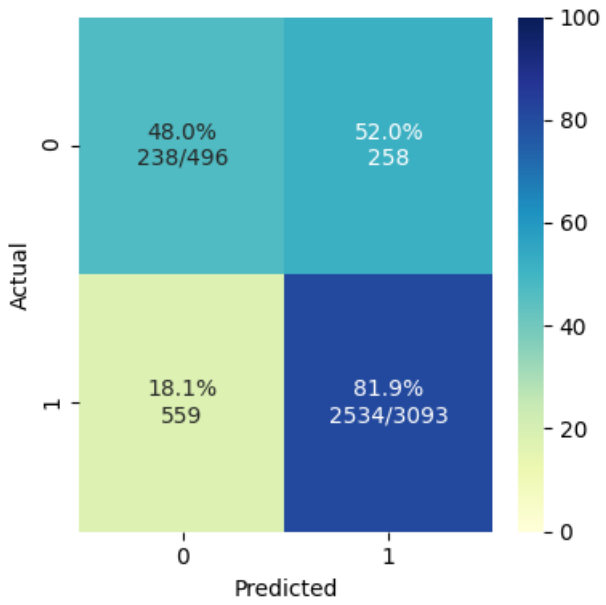


Figure 6.2: Confusion matrix depicting results for NN_1.

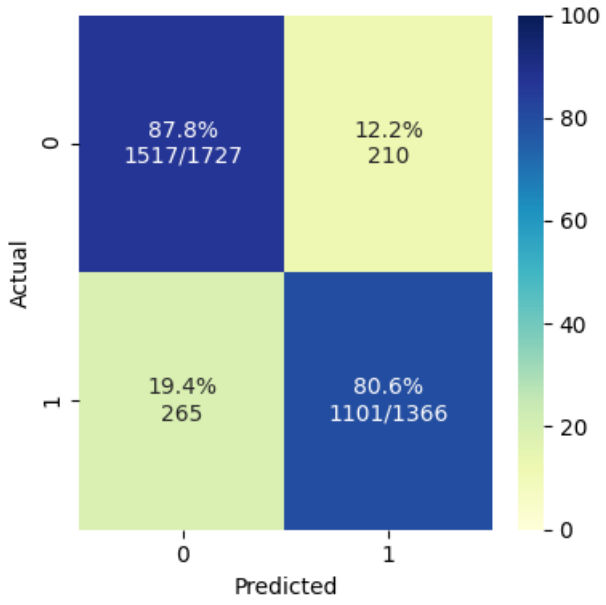


Figure 6.3: Confusion matrix depicting results for NN_2.

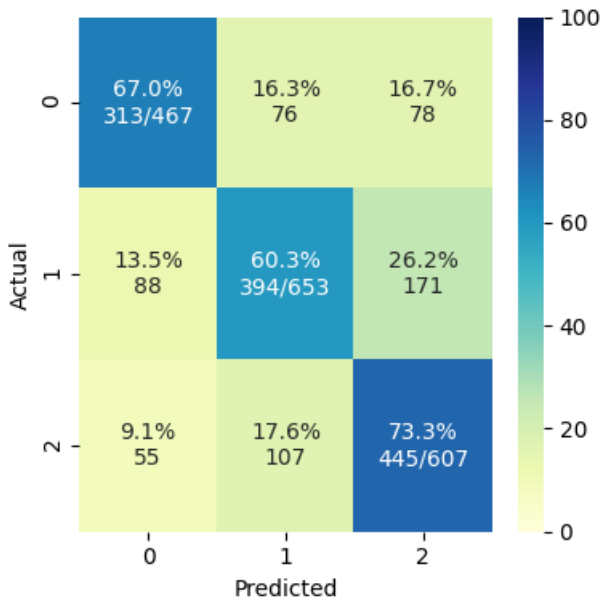


Figure 6.4: Confusion matrix depicting results for NN_3.

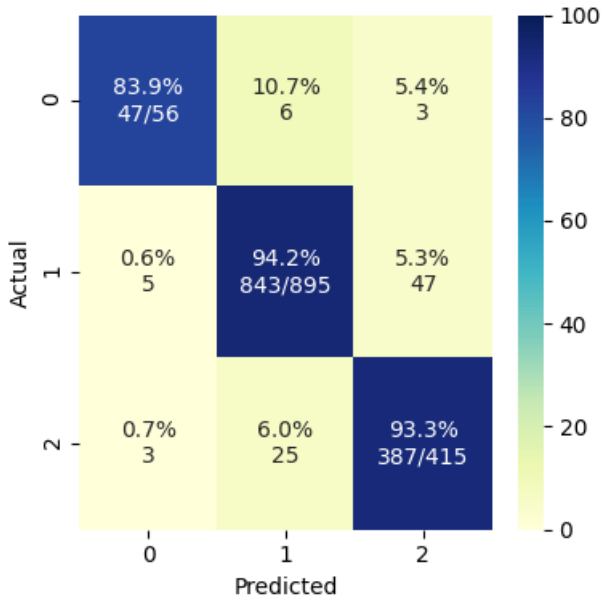


Figure 6.5: Confusion matrix depicting results for NN_4.

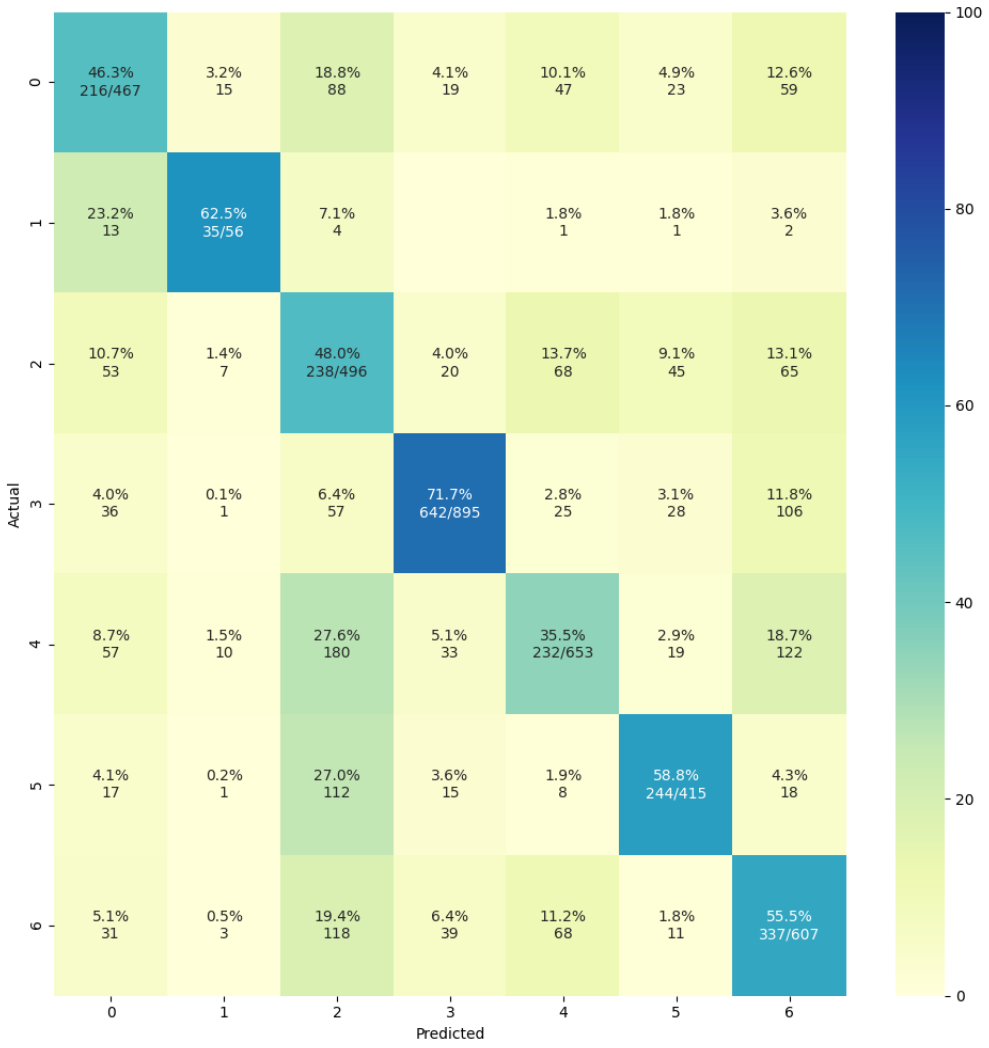


Figure 6.6: Confusion matrix depicting results for the overall decision tree architecture.

Table 6.6: Classification report for decision tree.

Class	Precision	Recall	f1-score	No. of examples
0	0.51	0.46	0.49	467
1	0.49	0.62	0.55	56
2	0.30	0.48	0.37	496
3	0.84	0.72	0.77	895
4	0.52	0.36	0.42	653
5	0.66	0.59	0.62	415
6	0.48	0.56	0.51	607
Average	0.54	0.54	0.53	358
Weighted average	0.57	0.54	0.55	3589

6.3 Discussion

We used two approaches to develop classifier responsible for recognizing human emotions - DT of CNNs and single CNN architectures. When using the DT, contrary to our expectations, overall performance was worse than in the case of single CNN. The main idea was to isolate problematic classes to achieve better results, but opposite turned out to be true. Implementing dedicated models just for the problematic classes, in our case class 2 (fear), did not help the overall performance. Despite that, accuracy just for the class 2 got slightly better. However, if wee look again at the tables 6.6 and 6.1, what can be observed, is that single CNN is better in every respect.

6.3.1 Analysis of the impact of sampling the data set

Following are the preliminary results using single CNN with not refined global parameters, just to demonstrate impact of the sampling procedure. When training on sampled data, accuracy of the model increased from around 53% to around 57%. Figures 6.7 and 6.8 show optimization progress during training with original data and with sampled data.

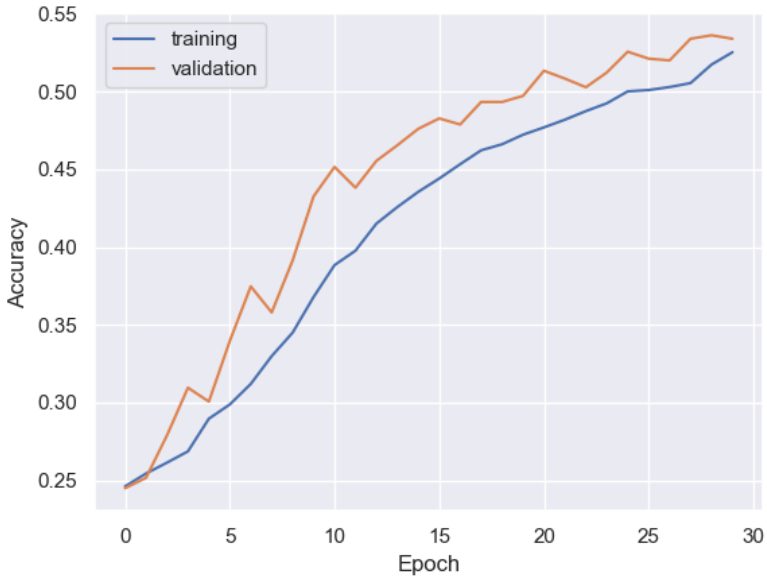


Figure 6.7: Optimization of the model on original training data.

Figure 6.8: Optimization of the model on sampled training data.

6.4 Model evaluation over an external data set

In the last section we compared two approaches we used to create a human emotion classifier. In this section, our goal is to analyze the performance of single CNN architecture (best performing model) over the external data set. In the following we will refer to the data set used for learning of the model (FER2013) as data set 1. Furthermore, data set used for evaluation (CK+) will be referred to as data set 2. The evaluation will be conducted in two steps:

1. Directly evaluating classifier on the new data set. We let our model, to predict emotions on input images from the data set 2 without touching the structure of our classifier. This way we can test if our model is able to generalize well in the new environment.

2. Using TL methods, we will preserve the knowledge learnt by the classifier on the data set 1 (in other words, connections and weights of the model), and reuse it for training on the data set 2. After training the new model with transferred knowledge, we can test it in the new environment with images of the data set 2.

But before we can go on with evaluating our model, one additional step is required. Our two selected data sets differ in one class. For this reason we have to perform mapping operation, in which we map images from data set 2 to corresponding class in data set 1. Because there is only one class that is different, one approach that could be pursued is to keep remaining 6 labels untouched and map only the different one. However, in real world, the label assigned to an image may not perfectly fit. This may be caused for instance by cultural differences, emotions can be represented differently in different cultures or regions. Since labels are assigned by humans, subjectivity can play a big part in this too. At last, images were taken in different environments, in the case of data set 1, images are captured in environment that is not controlled, as opposed to data set 2, where images were taken in lab environment. This can impose inconsistency regarding labels in the learning data sets. With that said, second approach that we tried, was to remap all images from the data set 2 to the new class corresponding to the data set 1.

6.4.1 Mapping procedure

To map labels from one data set to another, we need to compute *mass* center, or in other words, the most representative image for each class in the data set 1. This can be done in two ways. In first approach, we compute pixel by pixel difference between all images in one distinct class of emotion. After that, we find the image with lowest difference to every other image. This image will represent the *mass* center of that particular class. Then, we will follow this process for every emotion class. At the end, we end up with 7 images, each one representing mass center for one class of emotion. In some cases, pixel by pixel difference can be misleading. Thus, in our second approach we are representing each image with frequency histogram, which is much more comprehensive representation of image,

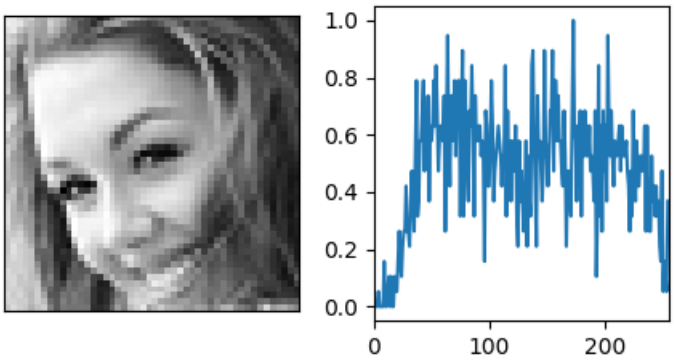


Figure 6.9: Example of image frequency histogram. On X axis there are individual pixel values. Intensity for individual pixel value is shown on Y axis.

an example of histogram is show on Figure 6.9. The process in this case is analogous, but instead of the pixel by pixel difference, we compute difference between histograms. We then once again extract the image with the lowest difference, that will represent the mass center of the specific emotion class.

6.4.2 Automatic label assignation

In both cases, we ended up with 7 images, each one representing the mass center of the one emotion class, labeled once again from 0 to 6. Finally, we need to assign a new label to images from data set 2. For each image in data set 2, we compute the difference between each mass center (pixel by pixel or histogram difference, based on the approach) and assign label such that, the difference between image and mass center is at minimum.

Results when remapping all 7 classes, were not optimal, as can be observed from confusion matrix 6.10. Images from one class are redistributed among all 7 classes almost randomly, which is not desirable. This may be caused by a fact, that there is only slight margin between the mass centers for respective classes, and thus only slight difference in the image representation can change the result of the mapping. When we kept the first 6 labels (identical for both data sets) untouched, the resulted mapping is much more clearer. Nevertheless, the results differ based on the approach. On Figure 6.11 we can see confusion matrix depicting mapping results when histogram difference was used. We can observe that the images were mostly remapped as happiness (in almost 50% of the cases). On the other hand, when using pixel by pixel difference, the dominant class is fear (with more than 50%), as shown on Figure 6.12.

Some of the images from data set 1 were containing a lot background noise, what in some cases produced undesirable outcome. We therefore incorporated face cropping, in order to reduce the noise and to focus on more relevant information contained in the image. Cropping was used only with the static labels method, since when remapping all 7 classes results were not desirable. Once

again, the outcome depends on the approach. On Figure 6.13, we can observe that the result is similar as when face cropping was not incorporated, shown on Figure 6.11. Mostly, images were remapped as happiness. In the case of pixel by pixel difference, the dominant classes is this time happiness (around 50%), as shown on Figure 6.14.

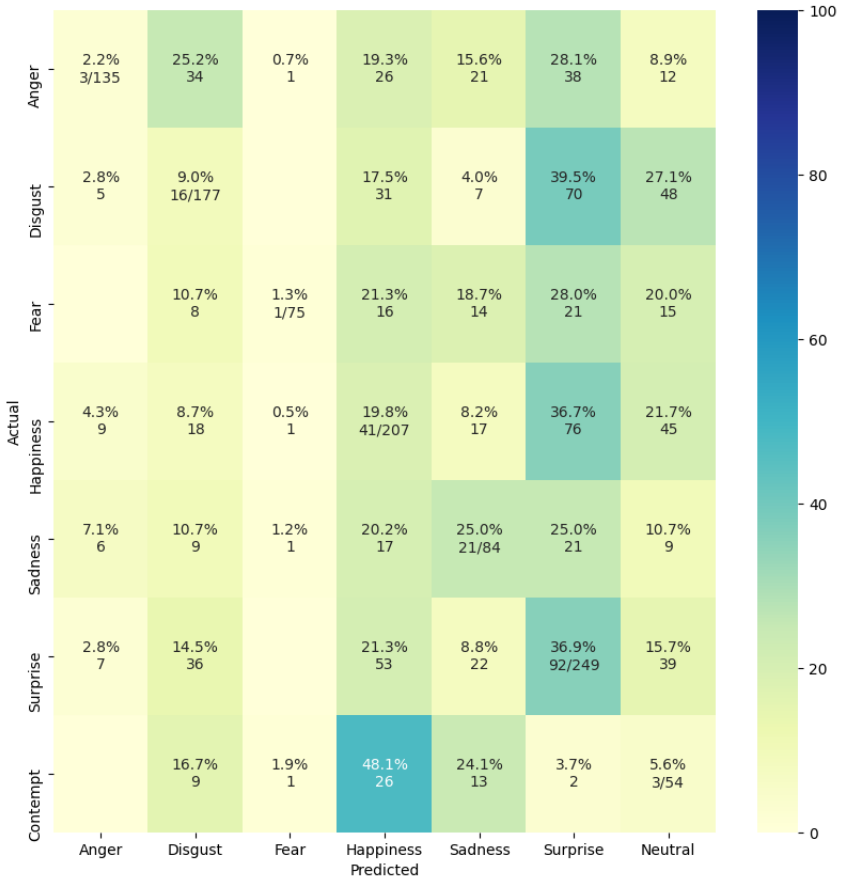


Figure 6.10: Confusion matrix depicting remapping of all 7 classes. On Y axis we have emotion labels from the data set 2. On the X axis, there are labels from data set 1, i.e. the predicted labels - since we are predicting to which class the image belongs.

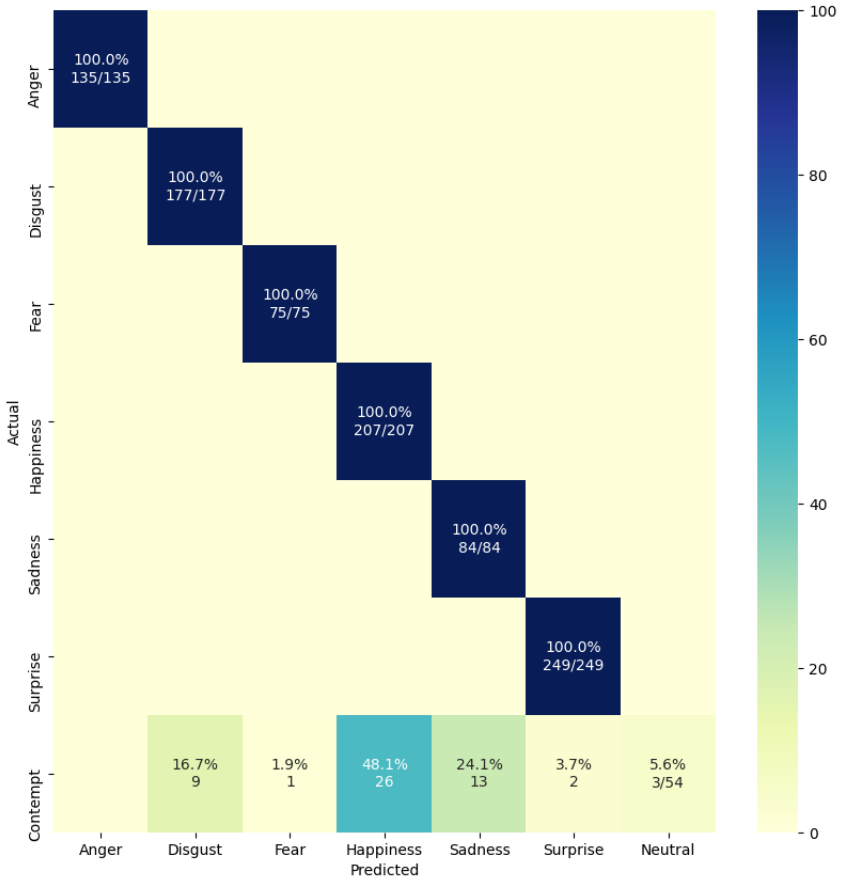


Figure 6.11: Confusion matrix depicting remapping of one class using histogram difference.

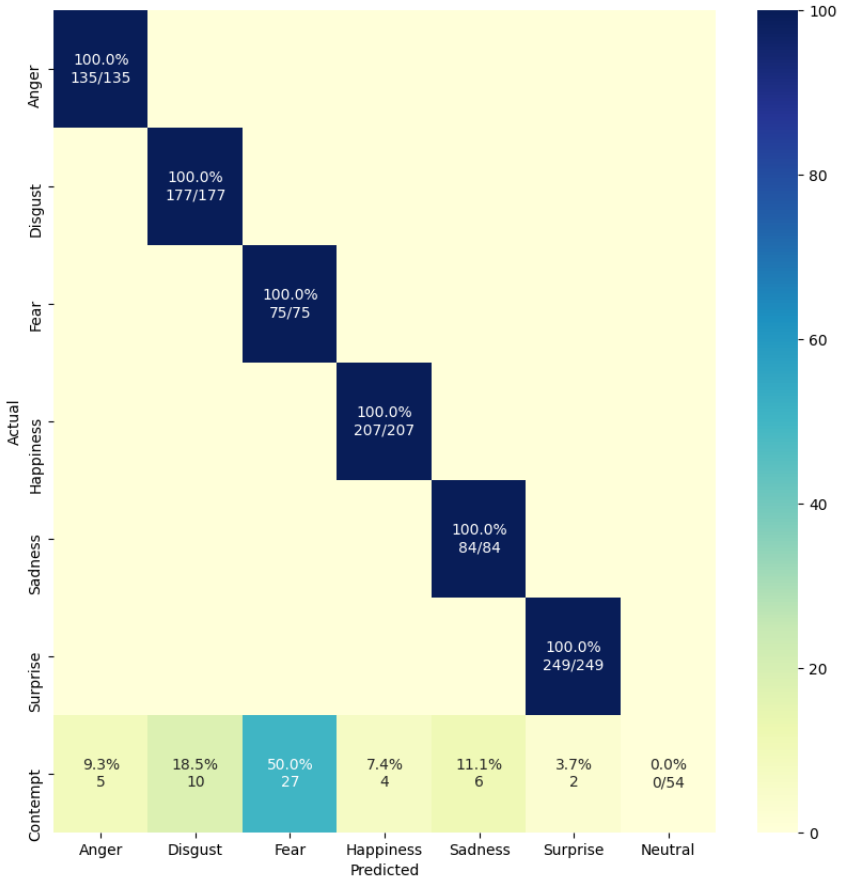


Figure 6.12: Confusion matrix depicting remapping of one class using pixel by pixel difference.

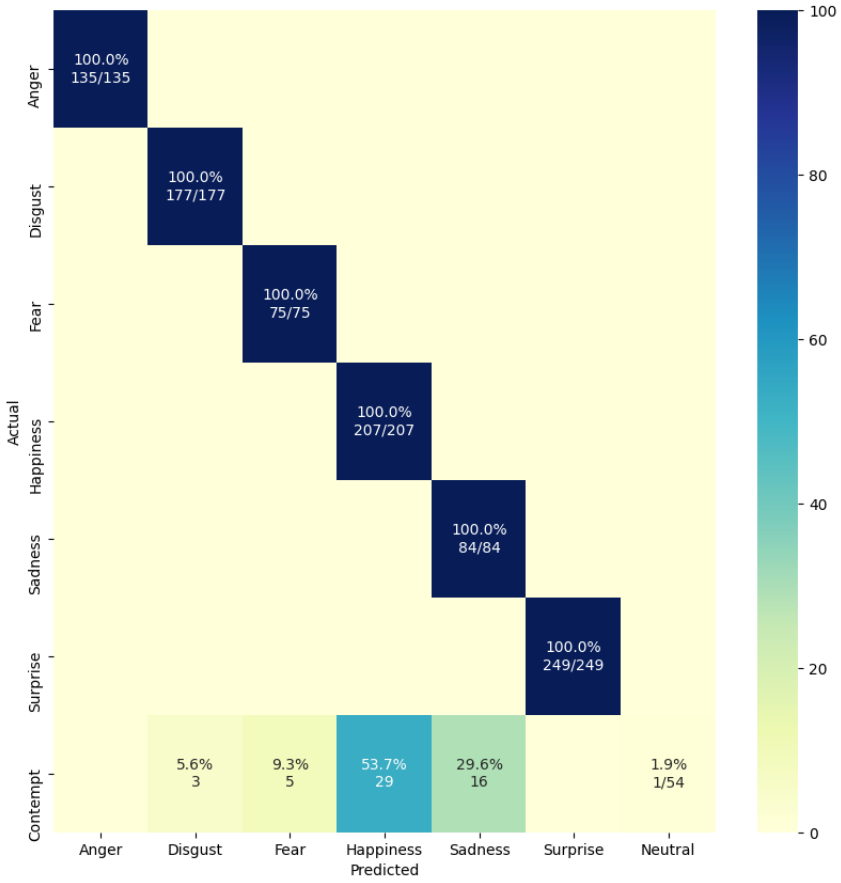


Figure 6.13: Confusion matrix depicting remapping of one class using histogram difference with face cropping incorporated.

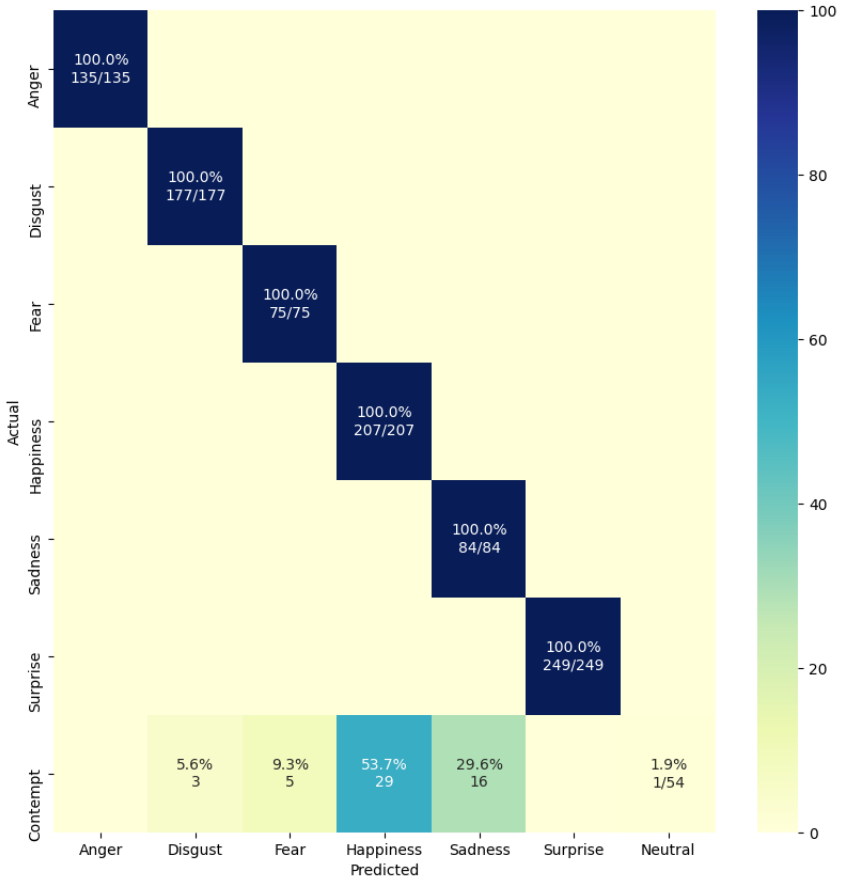


Figure 6.14: Confusion matrix depicting remapping only for one class using pixel by pixel difference with face cropping incorporated.

6.4.3 Direct evaluation

Based on mapping results, we decided to map all images from the class contempt in the data set 2 (the differently labeled class) as happiness, which was dominant class in most of the cases. In this section we present results of the direct evaluation of our classifier over the data set 2. We are evaluating the final single CNN model 6.1, since it yielded the best results. To perform evaluation, we consider the whole data set 2 as the testing data, with just one exception. We do not have any images from the neutral class (since data set 2 does not include it, and we used the mapping procedure), thus we have only 6 unique labels. However, our model is implemented to recognize 7 emotion classes. This means, that in some cases, it may predict some images as a neutral class. On

Figure 6.15 is confusion matrix depicting evaluating results. Last row (class 6 - neutral) is omitted, due to the reasons mentioned above. We can observe that the generalization abilities are reasonable. Class 2 (fear), similarly as for the original data set, has poor results with accuracy around 41%. Overall accuracy is 69%.

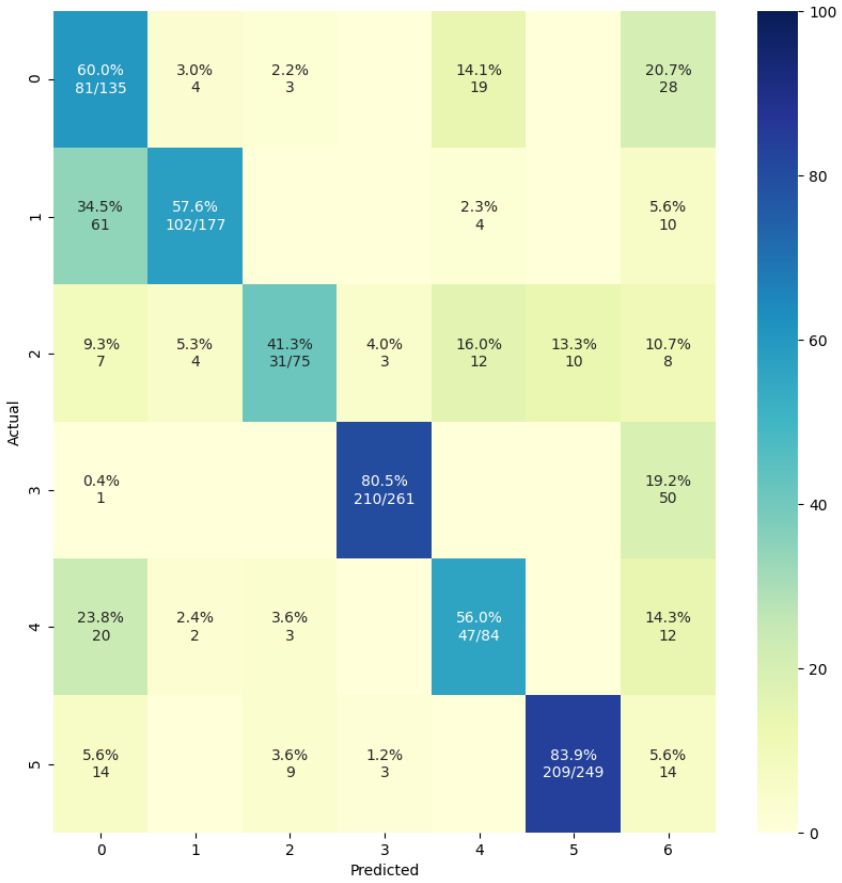


Figure 6.15: Confusion matrix evaluating the single convolutional neural network model over the external data set. Note that, we are omitting last row, since class 6 (contempt) was remapped as happiness. Therefore we have no images of class 6. However, there is still information present for the column 6, since the network was trained to classify images into 7 emotion classes.

6.4.4 Results applying transfer learning

We present two approaches used in order to apply TL, retraining only output layer, therefore not only preserving features in convolutional layer but also last fully-connected layer. Second approach

Table 6.7: Results for applying transfer learning, preserving only convolutional layers.

	5-fold	**8-fold**	**10-fold**
Mean Accuracy (%)	93.8	95.1	93.5

Table 6.8: Results for applying transfer learning, retraining only output layer.

	5-fold	**8-fold**	**10-fold**
Mean Accuracy (%)	82.8	85.3	84.6

is to retrain also last fully connected layer as well as output layer. Furthermore, both approaches are tested using whole data set, and also with reduced data set (only peak facial expression), this is described more in section 4.2. When the whole data set is used, we split 981 images into two selections, one for the training (80% of the data), and one for testing (20%). When we consider only peak facial expression (327 images), we convey procedure called K-fold cross-validation. In this method, data is divided into K groups, $K-1$ groups are used for the training and the 1 remaining group is used for the testing. Every group will be used once as the testing group, therefore training-testing process is repeated K times. We then extract the mean accuracy out of all K tries. In our work, we used 5, 8 and 10 as the values of K.

The results for the first approach on the whole data set are present on Figure 6.16. Accuracy for all classes is well above 70%, with the exception of class 2 (fear), which is once again the problematic class. Overall accuracy is 84%. Results using reduced data set are shown in Table 6.8, we can see that mean accuracy is around 84% for all values of K, which is similar as for the whole data set. When we conveyed the second approach on the whole data set, model achieved 100% accuracy. If we however, consider only peak facial expression and use K-fold cross-validation, accuracy decreases to around 94%, as shown on Table 6.7. This decrease is expected, since the training-testing process is conveyed K times, as opposed to the testing on whole data set, where this process is conveyed just once, therefore the results might not be proper. Nonetheless, we achieved state-of-the-art performance with regard to results shown in Table 4.3 on CK+ data set with the accuracy of 95% at best. We can overall conclude, that when we preserve only features learnt by convolutional layers (second approach), performance increases and our model is generalizing well.

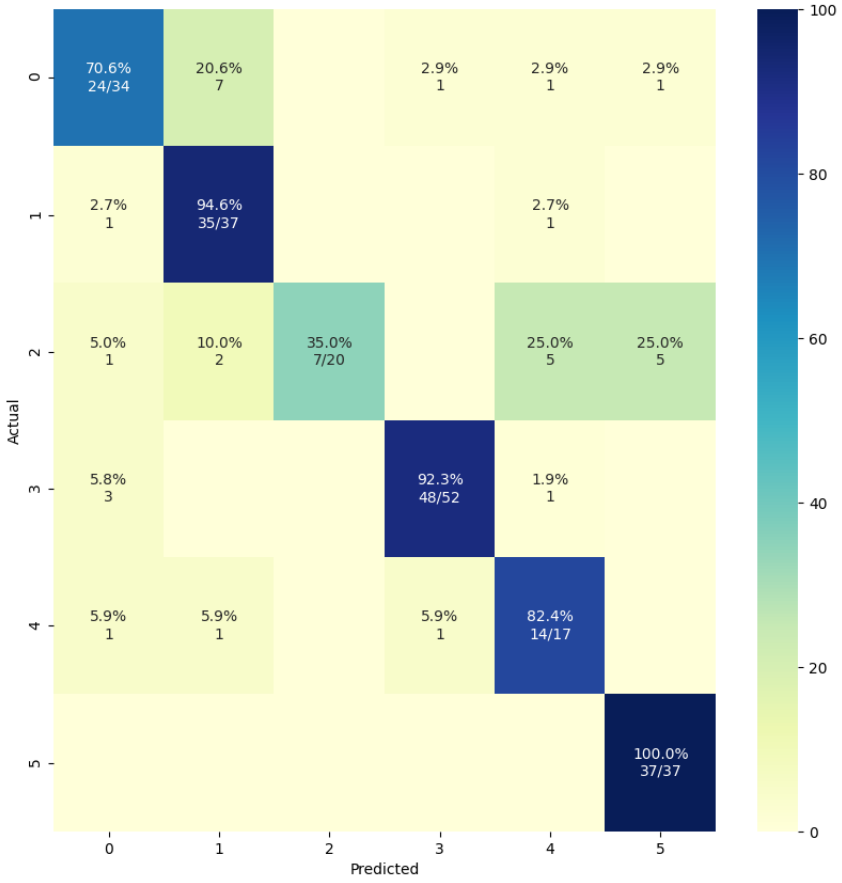


Figure 6.16: Confusion matrix depicting the application of transfer learning methodology, retraining only output layer.

Chapter 7

Conclusions

The goal of this work was to develop a system for recognizing emotions based on facial expressions in images. We utilized CNNs, a well-established computational model in the area of DL. We present different architectures based on a single CNN. Our final single CNN model is inspired by architecture of VGG network, which employs relatively deep network with many hidden layers, with the use of 3×3 convolutional filters. We managed to achieve state-of-the-art performance with accuracy of 66% on well-known FER2013 data set. We also utilized architecture based on Decision Tree, where the nodes are CNN models. DT however, contrary to our expectations, didn't achieve optimal results in comparison with single CNN architecture.

Furthermore, we conducted evaluation of presented single CNN architecture over external data set to test its generalization performance. For the purpose of evaluation we used another well-know FER data set, namely CK+. Since our selected data sets were not identical, i.e. they differ in one class label, we carried out mapping procedure, so as to assign label corresponding to FER2013 data set, to images from differently labeled class in CK+ data set (class contempt). In order to perform such mapping, we implemented algorithm to determine mass center per each class in FER2013 data set, utilizing histogram difference and pixel by pixel difference. Based on this we decided to map all images from class contempt as happiness. The evaluation was conducted in two distinct ways. At first we conveyed direct evaluation, where we consider CK+ data set as the testing data for our classifier. Our model achieved sub-optimal results with accuracy of 69%. The second approach in order to evaluate our architecture was to apply TL techniques to retain knowledge learnt by our model in the domain of FER2013 data set and reuse it in the domain of CK+ data set. We created a new model, that was utilizing this transferred knowledge. Model was then adapted to the new environment of CK+ data set, i.e. retrained with data from CK+ data set. The adaptation was done in 2 ways. At first we retrained only output layer, this model generalized well with accuracy of 84%. Second approach was to retain only convolution layers, and thus on top of output layer retrain also last fully-connected layer. This model scored accuracy of 95% at best, showing great generalization performance and achieving state-of-the-art performance on the CK+ data set.

Bibliography

1. LINDQUIST, Kristen A.; MACCORMACK, Jennifer K.; SHABLACK, Holly. The role of language in emotion: predictions from psychological constructionism. *Frontiers in Psychology*. 2015, vol. 6, p. 444. ISSN 1664-1078. Available from DOI: 10.3389/fpsyg.2015.00444.

2. BARRETT, Lisa Feldman. The Future of Psychology: Connecting Mind to Brain. *Perspectives on Psychological Science*. 2009, vol. 4, no. 4, pp. 326–339. Available from DOI: 10.1111/j.1745-6924.2009.01134.x. PMID: 19844601.

3. REISENZEIN, Rainer; STUDTMANN, Markus; HORSTMANN, Gernot. Coherence between Emotion and Facial Expression: Evidence from Laboratory Experiments. *Emotion Review*. 2013, vol. 5, no. 1, pp. 16–23. Available from DOI: 10.1177/1754073912457228.

4. EKMAN, P. Facial expression and emotion. *American Psychologist*. 1993, vol. 48, no. 4, pp. 384–392.

5. ZENG, Z.; PANTIC, M.; ROISMAN, G. I.; HUANG, T. S. A Survey of Affect Recognition Methods: Audio, Visual, and Spontaneous Expressions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2009, vol. 31, no. 1, pp. 39–58. Available from DOI: 10.1109/TPAMI.2008.52.

6. SHU, Lin; XIE, Jinyan; YANG, Mingyue; LI, Ziyi; LI, Zhenqi; LIAO, Dan; XU, Xiangmin; YANG, Xinyi. A Review of Emotion Recognition Using Physiological Signals. *Sensors*. 2018, vol. 18, no. 7. ISSN 1424-8220. Available from DOI: 10.3390/s18072074.

7. MARTINEZ, B.; VALSTAR, M. F.; JIANG, B.; PANTIC, M. Automatic Analysis of Facial Actions: A Survey. *IEEE Transactions on Affective Computing*. 2019, vol. 10, no. 3, pp. 325–347. Available from DOI: 10.1109/TAFFC.2017.2731763.

8. LI, Shan; DENG, Weihong. Deep Facial Expression Recognition: A Survey. *IEEE Transactions on Affective Computing*. 2020, pp. 1–1. ISSN 2371-9850. Available from DOI: 10.1109/taffc.2020.2981446.

9. LECUN, Yann; HAFFNER, Patrick; BOTTOU, Léon; BENGIO, Yoshua. Object Recognition with Gradient-Based Learning. In: *Shape, Contour and Grouping in Computer Vision*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 319–345. ISBN 978-3-540-46805-9. Available from DOI: 10.1007/3-540-46805-6_19.

10. SIMONYAN, Karen; ZISSERMAN, Andrew. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv preprint arXiv:1409.1556*. 2015.

11. GOODFELLOW, Ian; ERHAN, Dumitru; CARRIER, Pierre; COURVILLE, Aaron; MIRZA, Mehdi; HAMNER, Ben; CUKIERSKI, Will; TANG, Yichuan; THALER, David; LEE, Dong-Hyun; ZHOU, Yingbo; RAMAIAH, Chetan; FENG, Fangxiang; LI, Ruifan; WANG, Xiaojie; ATHANASAKIS, Dimitris; SHAWE-TAYLOR, John; MILAKOV, Maxim; PARK, John; BENGIO, Y. Challenges in Representation Learning: A Report on Three Machine Learning Contests. *Neural Networks*. 2013-07, vol. 64. ISBN 978-3-642-42050-4. Available from DOI: 10.1016/j.neunet.2014.09.005.

12. LUCEY, P.; COHN, J. F.; KANADE, T.; SARAGIH, J.; AMBADAR, Z.; MATTHEWS, I. The Extended Cohn-Kanade Dataset (CK+): A complete dataset for action unit and emotion-specified expression. In: *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Workshops*. 2010, pp. 94–101. Available from DOI: 10.1109/CVPRW.2010.5543262.

13. YU, Zhiding; ZHANG, Cha. Image Based Static Facial Expression Recognition with Multiple Deep Network Learning. In: *Proceedings of the 2015 ACM on International Conference on Multimodal Interaction*. New York, NY, USA: Association for Computing Machinery, 2015, pp. 435–442. ISBN 9781450339124. Available from DOI: 10.1145/2818346.2830595.

14. KIM, Bo-Kyeong; ROH, Jihyeon; DONG, Suh-Yeon; LEE, Soo-Young. Hierarchical committee of deep convolutional neural networks for robust facial expression recognition. *Journal on Multimodal User Interfaces*. 2016-01, vol. 10. Available from DOI: 10.1007/s12193-015-0209-0.

15. LI, Huihui; WEN, Guihua. Sample awareness-based personalized facial expression recognition. *Applied Intelligence*. 2019-08, vol. 49. Available from DOI: 10.1007/s10489-019-01427-2.

16. MENG, Z.; LIU, P.; CAI, J.; HAN, S.; TONG, Y. Identity-Aware Convolutional Neural Network for Facial Expression Recognition. In: *2017 12th IEEE International Conference on Automatic Face Gesture Recognition (FG 2017)*. 2017, pp. 558–565. Available from DOI: 10.1109/FG.2017.140.

17. LI, Ming; XU, Hao; HUANG, Xingchang; SONG, Zhanmei; LIU, Xiaolin; LI, Xin. Facial Expression Recognition with Identity and Emotion Joint Learning. *IEEE Transactions on Affective Computing*. 2018-11, vol. PP, pp. 1–1. Available from DOI: 10.1109/TAFFC.2018.2880201.

18. SZEGEDY, Christian; LIU, Wei; JIA, Yangqing; SERMANET, Pierre; REED, Scott; ANGUELOV, Dragomir; ERHAN, Dumitru; VANHOUCKE, Vincent; RABINOVICH, Andrew. Going deeper with convolutions. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 1–9. Available from DOI: 10.1109/CVPR.2015.7298594.

19. MOLLAHOSSEINI, Ali; CHAN, David; MAHOOR, Mohammad H. Going deeper in facial expression recognition using deep neural networks. *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*. 2016-03. ISBN 9781509006410. Available from DOI: 10.1109/wacv.2016.7477450.

20. YANAN GUO; DAPENG TAO; JUN YU; HAO XIONG; YAOTANG LI; DACHENG TAO. Deep Neural Networks with Relativity Learning for facial expression recognition. In: *2016 IEEE International Conference on Multimedia Expo Workshops (ICMEW)*. 2016, pp. 1–6. Available from DOI: 10.1109/ICMEW.2016.7574736.

21. HUNG, Jason C.; LIN, Kuan-Cheng; LAI, Nian-Xiang. Recognizing learning emotion based on convolutional neural networks and transfer learning. *Applied Soft Computing*. 2019, vol. 84, p. 105724. ISSN 1568-4946. Available from DOI: https://doi.org/10.1016/j.asoc.2019.105724.

22. MING, Z.; CHAZALON, J.; MUZZAMIL LUQMAN, M.; VISANI, M.; BURIE, J. FaceLiveNet: End-to-End Networks Combining Face Verification with Interactive Facial Expression-Based Liveness Detection. In: *2018 24th International Conference on Pattern Recognition (ICPR)*. 2018, pp. 3507–3512. Available from DOI: 10.1109/ICPR.2018.8545274.

23. LIU, Y.; CAO, Y.; LI, Y.; LIU, M.; SONG, R.; WANG, Y.; XU, Z.; MA, X. Facial expression recognition with PCA and LBP features extracting from active facial patches. In: *2016 IEEE International Conference on Real-time Computing and Robotics (RCAR)*. 2016, pp. 368–373. Available from DOI: 10.1109/RCAR.2016.7784056.

24. HAPPY, S. L.; ROUTRAY, A. Automatic facial expression recognition using features of salient facial patches. *IEEE Transactions on Affective Computing*. 2015, vol. 6, no. 1, pp. 1–12. Available from DOI: 10.1109/TAFFC.2014.2386334.

25. RYU, Byungyong; RIVERA, Adín Ramírez; KIM, J.; CHAE, O. Local Directional Ternary Pattern for Facial Expression Recognition. *IEEE Transactions on Image Processing*. 2017, vol. 26, pp. 6006–6018.

26. KOTSIA, I.; PITAS, I. Facial Expression Recognition in Image Sequences Using Geometric Deformation Features and Support Vector Machines. *IEEE Transactions on Image Processing*. 2007, vol. 16, no. 1, pp. 172–187. Available from DOI: 10.1109/TIP.2006.884954.

27. KOELSTRA, S.; PANTIC, M.; PATRAS, I. A Dynamic Texture-Based Approach to Recognition of Facial Actions and Their Temporal Models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 2010, vol. 32, no. 11, pp. 1940–1954. Available from DOI: 10.1109/TPAMI.2010.50.

28. HAYKIN, Simon. *Neural Networks and Learning Machines*. Pearson, 2009.

29. NIELSEN, Michael A. *Neural Networks and Deep Learning*. Determination Press, 2015. http://neuralnetworksanddeeplearning.com.

30. GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

31. WIKIPEDIA CONTRIBUTORS. *Convolution — Wikipedia, The Free Encyclopedia*. 2021. Available also from: https://en.wikipedia.org/w/index.php?title=Convolution&oldid=998195435. Accessed: 25-4-2021.

32. WU, Jianxin. *Introduction to convolutional neural networks*. 2017-05. Tech. rep. National Key Lab for Novel Software Technology. Nanjing University. China.

33. POLETAEV, I.; PERVUNIN, Konstantin; TOKAREV, M. *Artificial neural network for bubbles pattern recognition on the images - Scientific Figure on ResearchGate*. 2016-10. Available also from: https://www.researchgate.net/profile/Konstantin_Pervunin/publication/309487032/figure/fig2/AS:422116370718724@1477651793847/a-Illustration-of-the-operation-principle-of-the-convolution-kernel-convolutional-layer.png.

34. YOSINSKI, Jason; CLUNE, Jeff; BENGIO, Yoshua; LIPSON, Hod. How Transferable Are Features in Deep Neural Networks? In: *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*. Montreal, Canada: MIT Press, 2014, pp. 3320–3328. NIPS'14.

35. TEAM, Datahacker. *Edge detection*. Datahacker, 2020-12. Available also from: http://media5.datahacker.rs/2018/10/multiplication_slicice.png. Accessed: 25-4-2021.

36. WU, J. *The Lenna image and the effect of different convolution kernels*. 2017. Available also from: https://www.semanticscholar.org/paper/Introduction-to-Convolutional-Neural-Networks-Wu/450ca19932fcef1ca6d0442cbf52fec38fb9d1e5/figure/5. Accessed: 25-4-2021.

37. NG, Andrew; KATANFOROOSH, Kian; MOURRI, Y.B. *Convolutional Neural Networks*. Coursera, [n.d.]. Available also from: https://www.coursera.org/learn/convolutional-neural-networks. Accessed: 25-4-2021.

38. WIKIPEDIA CONTRIBUTORS. *Max pooling with a 2x2 filter and stride = 2*. 2021. Available also from: https://en.wikipedia.org/wiki/Convolutional_neural_network#/media/File:Max_pooling.png. Accessed: 15-4-2021.

39. KRIZHEVSKY, Alex; SUTSKEVER, Ilya; HINTON, Geoffrey. ImageNet Classification with Deep Convolutional Neural Networks. *Neural Information Processing Systems*. 2012-01, vol. 25. Available from DOI: 10.1145/3065386.

40. RUSSAKOVSKY, Olga; DENG, Jia; SU, Hao; KRAUSE, Jonathan; SATHEESH, Sanjeev; MA, Sean; HUANG, Zhiheng; KARPATHY, Andrej; KHOSLA, Aditya; BERNSTEIN, Michael; BERG, Alexander C.; FEI-FEI, Li. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*. 2015, vol. 115, no. 3, pp. 211–252. Available from DOI: 10.1007/s11263-015-0816-y.

41. LIN, M.; CHEN, Q.; YAN, S. Network In Network. *CoRR*. 2014, vol. abs/1312.4400.

42. BISHOP, Christopher M. *Neural Networks for Pattern Recognition*. Oxford University Press, 1995.

43. KINGMA, Diederik; BA, Jimmy. Adam: A Method for Stochastic Optimization. *International Conference on Learning Representations*. 2014-12.

44. TIELEMAN, Tijmen; HINTON, Geoffrey. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning*. 2012, vol. 4, no. 2, pp. 26–31.

45. THEANO DEVELOPMENT TEAM. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*. 2016-05, vol. abs/1605.02688. Available also from: http://arxiv.org/abs/1605.02688.

46. ABADI, Martin; BARHAM, Paul; CHEN, Jianmin; CHEN, Zhifeng; DAVIS, Andy; DEAN, Jeffrey; DEVIN, Matthieu; GHEMAWAT, Sanjay; IRVING, Geoffrey; ISARD, Michael; KUDLUR, Manjunath; LEVENBERG, Josh; MONGA, Rajat; MOORE, Sherry; MURRAY, Derek G.; STEINER, Benoit; TUCKER, Paul; VASUDEVAN, Vijay; WARDEN, Pete; WICKE, Martin; YU, Yuan; ZHENG, Xiaoqiang. TensorFlow: A system for large-scale machine learning. In: *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 2016, pp. 265–283. Available also from: https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf.

47. PASZKE, Adam; GROSS, Sam; MASSA, Francisco; LERER, Adam; BRADBURY, James; CHANAN, Gregory; KILLEEN, Trevor; LIN, Zeming; GIMELSHEIN, Natalia; ANTIGA, Luca; DESMAISON, Alban; KOPF, Andreas; YANG, Edward; DEVITO, Zachary; RAISON, Martin; TEJANI, Alykhan; CHILAMKURTHY, Sasank; STEINER, Benoit; FANG, Lu; BAI, Junjie; CHINTALA, Soumith. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In: WALLACH, H.; LAROCHELLE, H.; BEYGELZIMER, A.; D'ALCHÉ-BUC, F.; FOX, E.; GARNETT, R. (eds.). *Advances in Neural Information Processing Systems 32*. Curran Associates, Inc., 2019, pp. 8024–8035. Available also from: http://papers.neurips.

cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

48. CHOLLET, François et al. *Keras* [https://keras.io]. 2015.

49. O'MALLEY, Tom; BURSZTEIN, Elie; LONG, James; CHOLLET, François; JIN, Haifeng; INVERNIZZI, Luca, et al. *Keras Tuner* [https://github.com/keras-team/keras-tuner]. 2019.

50. LI, Lisha; JAMIESON, Kevin; DESALVO, Giulia; ROSTAMIZADEH, Afshin; TALWALKAR, Ameet. Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization. *J. Mach. Learn. Res.* 2017-01, vol. 18, no. 1, pp. 6765–6816. ISSN 1532-4435.

51. HASTIE, T.; TIBSHIRANI, R.; FRIEDMAN, J. *The elements of Statistical Learning*. New York, USA: Springer-Verlag, 2001. Spring series in statistics. ISBN 0387952845.

52. LIU, Weiyang; WEN, Yandong; YU, Zhiding; YANG, Meng. Large-Margin Softmax Loss for Convolutional Neural Networks. In: *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*. New York, NY, USA: JMLR.org, 2016, pp. 507–516. ICML'16.

53. SMITH, Leslie N. A disciplined approach to neural network hyper-parameters: Part 1 - learning rate, batch size, momentum, and weight decay. *ArXiv*. 2018, vol. abs/1803.09820.

54. NG, Andrew Y. Feature selection, L1 vs. L2 regularization, and rotational invariance. *Twenty-first international conference on Machine learning - ICML 04*. 2004. Available from DOI: 10.1145/1015330.1015435.

55. SOKOLOVA, Marina; LAPALME, Guy. A Systematic Analysis of Performance Measures for Classification Tasks. *Inf. Process. Manage.* 2009-07, vol. 45, no. 4, pp. 427–437. ISSN 0306-4573. Available from DOI: 10.1016/j.ipm.2009.03.002.

56. DIETTERICH, Tom. Overfitting and Undercomputing in Machine Learning. *ACM Comput. Surv.* 1995-09, vol. 27, no. 3, pp. 326–327. ISSN 0360-0300. Available from DOI: 10.1145/212094.212114.

57. IOFFE, Sergey; SZEGEDY, Christian. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In: *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*. Lille, France: JMLR.org, 2015, pp. 448–456. ICML'15.

58. PAN, Sinno Jialin; YANG, Qiang. A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering*. 2010, vol. 22, no. 10, pp. 1345–1359. Available from DOI: 10.1109/TKDE.2009.191.